| | |
|---|---|
| **Titre:** Title: | Improving the Energy Efficiency for Signal Processing and Machine Learning Algorithms Using Unreliable Memories |
| **Auteur:** Author: | Jonathan Kern |
| **Date:** | 2023 |
| **Type:** | Mémoire ou thèse / Dissertation or Thesis |
| **Référence:** Citation: | Kern, J. (2023). Improving the Energy Efficiency for Signal Processing and Machine Learning Algorithms Using Unreliable Memories [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/53445/ |

**POLYTECHNIQUE MONTRÉAL**
affiliée à l'Université de Montréal

ET

**IMT ATLANTIQUE**

# Improving the energy efficiency of signal processing and machine learning algorithms using unreliable memories

**JONATHAN KERN**
Département de génie électrique
**Polytechnique Montréal**
et
Département MEE
**IMT ATLANTIQUE**

Thèse en cotutelle présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie électrique

Mai 2023

**POLYTECHNIQUE MONTRÉAL**
affiliée à l'Université de Montréal

ET

**IMT ATLANTIQUE**

Cette thèse intitulée :

**Improving the energy efficiency of signal processing and machine learning algorithms using unreliable memories**

présentée par **Jonathan KERN**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

**James GOULET**, président
**François LEDUC-PRIMEAU**, membre et directeur de recherche
**Abdeldjalil AÏSSA EL BEY**, membre et codirecteur de recherche
**Elsa DUPRAZ**, membre et codirectrice de recherche
**Daniel MÉNARD**, membre
**Abdel-Ouahab BOUDRAA**, membre externe
**Kaoutar EL MAGHRAOUI**, membre externe

# ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my thesis advisors, Elsa Dupraz, François Leduc-Primeau and Abdeldjalil Aissa-El-Bey, for their guidance and support throughout my doctoral journey. Their insightful comments and constructive criticism, have been critical in shaping my ideas and improving the quality of this work. I am grateful for their mentorship, for the knowledge and skills I have acquired through our discussions and for their patience against the many administative hurdles of a jointly supervised thesis.

I would also like to extend my gratitude to my colleague Sebastien Henwood, whose collaboration was extremely valuable as well as their capacity for finding good paper names. His help was instrumental in pushing this thesis over the finish line, and I am grateful for his support and encouragement. My stay in Montreal would also not have been as enjoyable without him.

I am also grateful to my family for their support and encouragement throughout my academic journey. As the last member of my family to earn a doctorate, I am thrilled to finally join their ranks.

Lastly, I would like to thank my partner Miriam for her love, support, and most of all patience throughout this process.

# RÉSUMÉ

Le traitement du signal et les algorithmes d'apprentissage automatique sont désormais au cœur de notre monde numérique. Leur utilisation s'étend à de nombreux domaines tels que la navigation, les systèmes de communication numérique, ou encore le suivi médical. Cela a conduit à une forte augmentation de la consommation énergétique globale du secteur des technologies de l'information et de la communication. De plus, le nombre d'appareils à faible consommation d'énergie et fonctionnant sur batterie a également considérablement augmenté ces dernières années. Ces appareils nécessitent un haut niveau d'efficacité énergétique pour pouvoir exécuter les algorithmes les plus modernes. Pour répondre à ces problématiques, de nouvelles technologies ont été développées pour concevoir des mémoires plus économes en énergie, étant donné que la mémoire est responsable de la majorité de la consommation d'énergie d'un système électronique. Cependant, certaines de ces mémoires à faible consommation d'énergie ont pour inconvénient d'introduire un manque de fiabilité dans le système.

Dans cette thèse, nous étudions comment réduire la consommation d'énergie des algorithmes de traitement du signal et d'apprentissage automatique en utilisant des mémoires peu fiables et efficaces en énergie. Notre objectif est de minimiser l'effet du bruit sur l'algorithme étudié tout en essayant de réduire autant que possible la consommation énergétique de la mémoire. Ainsi, nous développons une méthodologie de réduction d'énergie applicable à différents algorithmes et technologies de mémoire. En utilisant notre modélisation des erreurs, nous effectuons d'abord une analyse théorique de l'effet des erreurs de mémoire sur les calculs. Nous proposons des équations liant la performance de l'algorithme étudié aux paramètres déterminant le niveau d'erreur et la consommation énergétique de la mémoire. Étant donné que ces paramètres contrôlent le compromis entre la consommation d'énergie du système et la performance de l'algorithme, nous formulons et résolvons un problème d'optimisation pour trouver l'ensemble optimal de paramètres qui minimise la consommation d'énergie de la mémoire tout en satisfaisant une contrainte de performance.

Nous appliquons d'abord cette méthodologie aux filtres de Kalman quantifiés. Nous considérons l'utilisation d'une mémoire dont la tension d'alimentation peut être réduite, au prix d'une augmentation du taux d'erreur sur les bits. Nous considérons que la quantification et les mémoires non fiables introduisent des erreurs dans les calculs, et nous développons un modèle de propagation des erreurs qui prend en compte ces deux sources d'erreurs. En plus de fournir des équations de filtre de Kalman actualisées, le modèle d'erreur proposé prédit avec précision la covariance de l'erreur d'estimation et donne une relation entre la performance

du filtre et sa consommation d'énergie, en fonction du niveau de bruit dans les mémoires. Nous introduisons ensuite deux méthodes d'optimisation pour minimiser la consommation d'énergie de la mémoire en fonction de la performance d'estimation souhaitée du filtre. La première méthode calcule les niveaux d'énergie optimaux alloués à chaque banque de mémoire individuellement, et la seconde optimise l'allocation d'énergie par groupes de banques de mémoire. Des simulations montrent une forte correspondance entre l'analyse théorique et les résultats expérimentaux. De plus, elles démontrent une réduction importante de la consommation d'énergie de plus de 50%.

Dans la deuxième partie de cette thèse, nous nous concentrons sur les réseaux de neurones profonds implémentés à l'aide de matrices de memristors. Les memristors permettent le calcul en mémoire, un domaine émergent qui peut être exploité par les réseaux de neurones lors de l'inférence pour réduire leur empreinte énergétique. Cependant, de tels gains en efficacité énergétique se font au prix de l'ajout d'un bruit sur les résultats des calculs effectués par les memristors. Dans ce travail, nous introduisons une méthode théorique pour estimer l'erreur quadratique moyenne d'un réseau de neurones implémenté sur memristors. Nous proposons une implémentation logicielle efficace de cette méthode qui s'avère être plus rapide de plusieurs ordres de grandeur que des simulations de Monte-Carlo pour prédire la performance. De plus, nous étudions deux techniques différentes pour configurer les calculs des couches convolutionnelles sur une implémentation de memristors et nous comparons leur impact relatif sur l'erreur quadratique moyenne et son temps de calcul. La précision de l'analyse proposée est d'abord évaluée sur un problème de régression simple, puis sur une tâche de classification plus complexe avec un réseau capable d'atteindre une grande précision sur le jeu de données CIFAR-10, ce qui montre que notre méthode est efficace par rapport aux réseaux de neurones modernes. Notre méthode est ensuite utilisée pour effectuer une optimisation heuristique de la valeur de conductance maximale du memristor afin de minimiser la consommation d'énergie.

Ainsi, dans cette thèse, nous avons présenté une méthodologie pour réduire la consommation d'énergie d'algorithmes utilisant une mémoire non fiable. Cette méthodologie pourrait être adapté pour être utilisé par d'autres algorithmes de traitement du signal ou d'apprentissage automatique, ou avec d'autres modèles de mémoire.

# ABSTRACT

Signal processing and machine learning algorithms are now at the central stage of our digital world, from navigation to digital communication, or even health monitoring. This has led to a strong increase in the global energy consumption of the Information and Communication Technology sector. The number of low power battery constrained devices has also drastically increased in the recent years. These devices require a high level of energy efficiency to be able to run state-of-the-art algorithms. To answer this need, new technologies have been developed to design more energy efficient memories, given that the memory is responsible for the majority of the energy consumption of an electronic system. However some of these energy-efficient memories come with the trade-off that they introduce unreliability in the system.

In this thesis we study how to reduce the energy consumption of signal processing and machine learning algorithms by using unreliable, energy-efficient memories. Our objective is to minimize the effect of noise onto the target algorithm while trying to reduce as much as possible the energy consumption of the memory. In this regards, we develop an energy-reduction methodology applicable to different algorithms and memory technologies. Based on our knowledge of the error model, we first carry out a theoretical analysis of the effect of the memory errors onto the computations. We propose equations linking the performance of the studied algorithm to the parameters directing the error level and energy consumption of the memory. Given that these parameters control the trade-off between energy consumption of the system and performance of the algorithm, we formulate and solve an optimization problem to find the optimal set of parameters that minimize the energy usage of the memory while satisfying a performance constraint.

We first apply this methodology to quantized Kalman filters. We consider using a voltage-scaled memory, for which the supply voltage of the memory can be reduced, at the price of an increased Bit-Error-Rate. We consider that both the quantization and the unreliable memories introduce errors in the computations, and we develop an error propagation model that takes into account these two sources of errors. In addition to providing updated Kalman filter equations, the proposed error model accurately predicts the covariance of the estimation error and gives a relation between the performance of the filter and its energy consumption, depending on the noise level in the memories. We then introduce two optimization methods to minimize the memory energy consumption under the desired estimation performance of the filter. The first method computes the optimal energy levels allocated to each memory

bank individually, and the second one optimizes the energy allocation per groups of memory banks. Simulations show a close match between the theoretical analysis and experimental results. Furthermore, they demonstrate an important reduction in energy consumption of more than 50%.

In the second part of this thesis, we focus on Deep Neural Networks (DNN) implemented using memristor crossbars. Memristors allow for in-memory computing, an emerging field which may be leveraged by DNN accelerators to reduce energy footprint. However, such gains in energy efficiency come at the cost of noise on the computation results due to the analog nature of memristors. In this work, we introduce a theoretical framework to estimate the Mean Squared Error (MSE) of a memristor-based DNN. We propose an efficient software implementation of this framework which is shown to be orders of magnitude faster than using Monte-Carlo simulations to predict the performance. Additionally, we study two different techniques for mapping convolutional layers to memristors and compare their relative impact on the MSE and its computation time. The accuracy of the proposed analysis is first evaluated on a simple regression problem, and then on a more complex classification task with a network capable of achieving high accuracy on the CIFAR-10 dataset, which shows that our method is efficient over practical up-to-date DNNs. The proposed framework is then used to perform a heuristic optimization of the memristor maximal conductance value so as to minimize the energy usage.

Thus, in this thesis we presented a framework for reducing the energy consumption of algorithms using an unreliable memory. This framework could be adapted to be used by other signal processing or machine learning algorithms, or to consider other memory models.

# CONDENSÉ EN FRANÇAIS

## Introduction

Avec l'apparition de nouvelles technologies et applications, la consommation énergétique des systèmes électroniques a considérablement augmenté au fil des ans. Cette augmentation significative s'explique à la fois par le nombre croissant de systèmes électroniques et par l'augmentation des besoins énergétiques des systèmes numériques et de leurs applications. La part du secteur des technologies de l'information et de la communication (TIC) dans la consommation mondiale d'électricité est passée de 3,9% en 2007 [1] à 7% en 2020 [2]. En outre, selon le rapport [2], l'empreinte énergétique mondiale du secteur des TIC pourrait doubler d'ici à 2030.

En particulier, le nombre de petits appareils embarqués a également fortement augmenté. Le nombre de dispositifs faisant partie de ce que l'on appelle l'Internet des objets a été estimé à 9 milliards en 2017 et devrait atteindre plus de 60 milliards d'ici 2025 [3]. La plupart de ces appareils fonctionneront avec de petites batteries ou par récolte d'énergie [4]. Cela inclut les dispositifs biomédicaux [5, 6] et d'autres types de capteurs intelligents [7, 8]. Il existe donc un besoin croissant de dispositifs embarqués à très faible consommation d'énergie capables d'exécuter divers algorithmes de traitement du signal avec un budget énergétique très contraint.

La loi de Moore énoncée en 1965 prévoyait que le nombre de transistors sur une puce electronique allait doubler tous les deux ans. L'augmentation de la densité des transistors a permis d'améliorer significativement et de manière continue l'efficacité énergétique des systèmes électroniques. Cependant, au cours des dernières années, l'efficacité énergétique de ces systèmes basée sur la loi de Moore a commencé à stagner [9] car nous commençons à atteindre des limites physiques, par exemple sur la taille des transistors en technologie CMOS. Il est donc essentiel d'améliorer l'efficacité énergétique des algorithmes de traitement du signal (TS) et d'apprentissage automatique (AA), et en particulier des réseaux de neurones profonds (RNP) qui sont maintenant largement utilisés dans divers domaines.

Il est maintenant largement reconnu que les mémoires sont les composants représentants la plus grande part de la consommation d'énergie d'un dispositif électronique. En effet, la mémoire est responsable de plus de la moitié de l'utilisation totale de l'énergie d'une puce de CPU [10]. Pour une simple opération arithmétique, l'accès à la mémoire peut coûter entre deux et trois ordres de grandeur d'énergie de plus que l'opération en elle même [11]. Par

conséquent, dans cette thèse, nous proposons de réduire la consommation d'énergie des systèmes electroniques en nous concentrant sur l'utilisation des mémoires. Ceci devrait favoriser l'émergence de nouvelles applications capables d'exécuter des algorithmes de TS et AA sur des systèmes à forte contrainte énergétique avec un haut niveau de performance.

Les méthodes existantes de réduction de la consommation d'énergie peuvent être divisées en deux grandes catégories. D'une part, les modifications logicielles consistent à modifier les algorithmes pour réduire leur consommation d'énergie. D'autre part, les modifications matérielles, consistent à optimiser divers paramètres de conception dans l'implémentation matérielle, tels que des changements dans l'architecture des composants ou l'utilisation de différentes technologies de composants. Enfin, la meilleure solution est souvent une approche hybride, avec une conception économe en énergie à la fois de l'algorithme et du matériel.

Dans cette thèse, nous proposons des modifications algorithmiques innovantes et adaptées à des solutions matérielles émergentes pour améliorer l'efficacité énergétique des algorithmes de TS et AA. Nous nous concentrons sur les nouvelles technologies de mémoire qui peuvent avoir une consommation d'énergie plus faible qu'une mémoire SRAM standard. Dans notre cas, nous étudions dans un premier temps le cas d'une SRAM dont la tension d'alimentation peut être ajustée pour réaliser des gains en énergie, puis dans un second temps, nous étudions les architectures de calcul en mémoires résistives utilisant des memristors. Ces deux types de mémoire représentent des technologies émergentes différentes qui ont connu un intérêt croissant au cours des dernières décennies en raison de leur potentiel à diminuer la part de la mémoire dans la consommation énergétique totale d'un système électronique. Cependant, la manière dont chacun de ces systèmes permet de réduire la consommation d'énergie d'un dispositif électronique est très différente.

En effet, une SRAM à tension réduite fonctionne de manière similaire à un système de mémoire classique, à l'exception de la consommation d'énergie des accès en lecture qui peut être diminuée. Alors qu'une mémoire résistive évite presque totalement le transfert de données entre la mémoire et l'unité de traitement en effectuant une partie importante des opérations de calcul directement dans la mémoire. Cependant, ces deux technologies possèdent un trait commun, à savoir que la réduction d'énergie qu'elles permettent se fait au prix de l'introduction d'erreurs dans les valeurs stockées. Ces erreurs peuvent être considérées comme du bruit introduit dans les calculs effectués par un dispositif utilisant ces mémoires non fiables.

Dans notre travail, nous étudions comment l'introduction d'erreurs dans les calculs effectués par les mémoires non fiables peut affecter certains algorithmes de TS et AA. Il est important de noter que l'effet des erreurs de calcul sur les performances dépend de l'algorithme considéré. Certains d'entre eux sont intrinsèquement robustes aux erreurs, et d'autres peuvent

être modifiés pour améliorer leur robustesse. Cependant, ces techniques d'amélioration de la robustesse consomment également de l'énergie. Par conséquent, pour un algorithme implémenté sur un certain système non fiable, nous pouvons définir un problème d'optimisation qui consiste à minimiser la consommation d'énergie du système tout en garantissant un certain niveau de performance malgré les erreurs de calcul. Dans ce travail, pour plusieurs algorithmes de TS et AA d'intérêt, nous cherchons à identifier et à résoudre un tel problème d'optimisation, en suivant une approche en trois étapes :

1. Fournir une analyse théorique de l'influence du bruit causé par la mémoire non fiable sur la performance de l'algorithme. Pour mesurer son impact, nous utilisons la mesure de l'erreur quadratique moyenne (EQM) entre la sortie correcte de l'algorithme et sa sortie bruitée lorsqu'il utilise la mémoire non fiable.

2. Exprimer à la fois l'EQM et la consommation d'énergie de la mémoire en fonction du niveau de bruit de la mémoire et d'autres paramètres de l'algorithme étudié.

3. Exprimer et résoudre le problème d'optimisation lié au compromis entre les performances de l'algorithme et la consommation d'énergie du système.

Dans cette thèse, nous appliquons ces objectifs à deux méthodes, l'une issue du domaine du TS, et l'autre du domaine de l'AA. Nous considérons d'abord le filtrage de Kalman et ensuite les RNP, tous deux implémentés à partir de matériel non fiable.

Bien que le filtrage de Kalman ait été introduit il y a plus de 60 ans, il est toujours largement utilisé pour une variété d'applications, en particulier pour le guidage et la navigation. En tant que tel, il est souvent mis en œuvre sur du matériel aux ressources limitées. Il est donc indispensable d'améliorer son efficacité énergétique. Les RNPs, quant à eux, ont connu un intérêt croissant plus récemment. Ils peuvent également être utiles pour les petites applications à forte contrainte énergétique, ainsi que pour les applications plus importantes nécessitant un grand nombre de paramètres. Ces deux algorithmes sont conçus pour traiter des données bruitées, ce qui en fait des candidats de choix pour traiter des perturbations dans leurs calculs. En outre, ils sont tous deux basés sur le calcul itératif, où l'entrée d'une étape de calcul utilise la sortie de l'étape précédente. Cela signifie que dans les deux cas, les erreurs introduites à une étape peuvent se propager et s'accumuler dans les étapes de calcul suivantes, ce qui rend encore plus importante la nécessité de les compenser. En effet, le filtre de Kalman consiste en des estimations successives à des instants successifs, tandis qu'un RNP est construit à partir d'un grand nombre de couches successives. Par conséquent, de telles similitudes entre ces deux algorithmes peuvent conduire à une méthodologie similaire pour

améliorer leur efficacité énergétique. Cependant, un filtre de Kalman standard est composé uniquement d'opérations linéaires, alors qu'un RNP présente également des non-linéarités introduites par ses fonctions d'activation. C'est pourquoi, dans ce qui suit, nous étudions l'implémentation d'un filtre de Kalman sur des SRAMs à tension réduite, tandis que nous étudions des RNP implementés dans des unités de calcul en mémoire construirtes à partir de matrices de memristors. Cela conduit à des modèles d'erreur différents dans chaque cas.

**Filtre de Kalman utilisant des mémoires non fiables**

Dans la première partie de cette thèse, nous étudions les filtre de Kalman [12] quantifiés et stockés sur une mémoire dont la tension d'alimentation peut être réduite pour la rendre plus efficace en énergie. Cependant, la réduction de la tension d'entrée de la mémoire augmente sa probabilité d'erreur, c'est à dire la probabilité qu'un bit soit inversé. Dans le modèle que l'on considère, la probabilité d'erreur par bit est inversement proportionelle à l'inverse de l'exponentielle de l'énergie fournie à la mémoire [13].

De plus, nous prenons aussi en compte que le filtre de Kalman est quantifié. Ainsi, il est possible de réduire la consommation d'énergie du filtre en réduisant son nombre de bits de quantification. Nous utilisons un modèle de représentation en virgule fixe, avec une quantification uniforme [14]. Dans notre cas, l'erreur de quantification peut être considérée comme un bruit blanc suivant une loi uniforme, de variance dépendant uniquement de la résolution de quantification.

Le filtre de Kalman a pour but de minimiser l'erreur quadratique moyenne entre les vraies valeurs d'état du système et son estimation. A chaque étape, le filtre réalise l'estimation et calcule aussi la covariance de l'erreur d'estimation. En utilisant les modèles définis précédemment, l'objectif est de calculer des versions mises à jours des équations du filtre de Kalman prenant en compte les nouvelles sources d'erreurs. Pour cela, nous étudions comment les erreurs se propagent dans le filtre de Kalman. L'erreur totale à une étape du filtre peut être décomposée en 2 termes : les erreurs venant des étapes précédentes et l'erreur ajoutée à l'étape actuelle, en raison de la quantification et des erreurs dans la mémoire. Nous exprimons d'abord séparément ces deux termes, avant de les combiner pour exprimer l'erreur totale sur l'estimation, en fonction de l'erreur à l'instant précédent, de la résolution de la quantification et de l'énergie fournie à la mémoire. En utilisant cette équation, il est alors possible de calculer la nouvelle matrice de covariance de l'erreur d'estimation qui pourra être utilisé dans la nouvelle version du filtre de Kalman.

De plus cette matrice de covariance peut nous servir de critère de performance du filtre avec lequel il est possible d'optimiser la consommation d'énergie de la mémoire. Ainsi, nous formulons deux problèmes d'optimisation, pour lesquels les paramètres d'optimisations sont le nombres de bits et l'énergie fournie à chaque banque de mémoire. Dans le premier, on optimise conjointement le nombre de bit, et les niveaux d'énergie et l'on considère qu'il est possible d'avoir autant de banques de mémoire que de nombres de bits de quantification. Dans le deuxième, le nombre de bits est fixés mais on considère que le nombre de niveaux d'énergie possible est inférieur au nombre de bits de quantifications. Il faut alors trouver à quels valeurs doivent être fixés chacun des niveaux, et aussi à quel niveau d'énergie chaque bit doit être assigné. Pour le premier problème, nous proposons une solution analytique pour calculer les meilleurs niveaux d'énergie pour chaque bits lorsque le nombre de bits de quantification est fixé. Il est alors possible de calculer cette solution pour une valeur maximum d'EQM souhaitée en employant un algorithme de *water-filling* [15]. Nous présentons un algorithme calculant cette solution analytique pour chaque valeur de nombre de bits de quantification afin de trouver le meilleur nombre de niveaux et leurs valeurs. Dans le cas du deuxième problème, nous formulons une solution analytique similaire à celle du problème 1 dans le cas où chaque bit est déjà assigné à un des niveaux d'énergie possible et où l'on cherche quelles sont les valeurs optimales de ces niveaux d'énergie. Cette solution doit ensuite être calculée pour chaque allocation de bit possible afin de trouver celle qui minimise l'énergie tout en respectant la contrainte de performance.

Nous présentons des résultats expérimentaux sur deux problèmes : un simple problème de suivi où nous estimons la position et vitesse d'un objet, ainsi qu'un plus gros problème de dimension 20 avec un modèle de transition d'état qui effectue un déplacement des entrées de l'état vers l'état suivant à chaque itération. Nos équations théoriques sont comparées à des simulations de Monte-Carlo ce qui permet de montrer que nos équations prédisent avec une très bonne précision la variance de l'erreur d'estimation. Ces résultats sont vérfiés pour différentes valeurs de niveaux de quantification ainsi que différents niveaux d'erreurs de la mémoire. Si le nombre de bits ou l'énergie fournie est trop faible, alors l'erreur de quantification ou le bruit de la mémoire domineront l'erreur totale d'estimation. On constate cependant qu'il existe un nombre minimal de bits à partir duquel, avec suffisamment d'énergie, il sera possible d'atteindre la variance minimale possible de l'erreur d'estimation.

Dans un deuxième temps, nous présentons des résultats expérimentaux provenant des solutions aux problèmes d'optimisation décrits précédemment. Nous montrons qu'utiliser une répartition d'énergie calculée avec notre solution permet de réduire la consommation d'énergie de 56% par rapport à une allocation d'énergie uniforme. De plus, nos expériences sur le deuxième problème d'optimisation permettent de montrer que seuls sept niveaux d'énergie

sont suffisants pour atteindre 95% du gain énergétique maximal qui a été obtenu dans le premier problème d'optimisation avec 20 bits de quantifications.

**Réseaux de neurones profonds utilisant des memristors**

Dans la deuxième partie de cette thèse, nous étudions les RNP implémentés en utilisant des unités de calcul en mémoire construites à partir de memristors [16]. Les memristors sont une technologie émergente pouvant être utilisé comme une forme de mémoire non-volatile très efficace en énergie [17]. Les memristors ont des niveaux de conductance variables, ce qui leur permet de stocker de l'information De plus, il est possible de réaliser des multiplications entre vecteur et matrices directement dans des matrices de memristors ce qui permet de réduire considérablement l'énergie nécessaire au transfert de données entre la mémoire et l'unité de calcul. Cependant, plusieurs sources d'erreurs tel que la variabilité entre les systèmes ou des courants parasites font que les calculs effectués sur memristors peuvent être considéré comme non fiables. Dans ce travail, nous considérons donc que les valeurs stockées par les memristor subissent un bruit.

Dans le cadre de ce travail, nous nous intéressons donc à l'implémentation de réseaux de neurones à l'aide de matrices de memristors. Nous considérons que les couches linéaires ainsi que les couches de convolutions exécutent leurs calculs sur des memristors bruités et que les autres opérations du réseau tel que les couches de *pooling* ou les fonctions d'activations sont effectuées sur un circuit CMOS fiable. Afin de transférer les poids des couches de réseaux de neurones sur les matrices de memristors, il est nécessaire de les multiplier par un facteur d'échelle. En effet, les valeurs de conductance doivent être comprises dans un intervalle de valeurs de conductances physiquement possibles. Cependant, en réduisant le facteur d'échelle, il est possible de réduire encore plus la conductance maximale stockée par les memristors ce qui permet de réduire leurs consommations d'énergie.

Pour transférer les opérations des couches de convolutions sur des calculs fait par les memristors, nous étudions deux possibilités. Dans la première, les couches de convolution sont d'abord transformée en couche linéaire, ce qui leur donne une matrice de poids très grande mais très creuse. De cette manière, le calcul d'une couche peut être effectué en un seul produit matrice-vecteur ce qui permet d'éviter d'ajouter de la corrélation entre les bruit de deux sorties différentes. Mais cela requiert une très grande matrice de memristor ce qui est peu pratique. La deuxième méthode d'implémentation de la convolution requiert de transférer les poids des kernels de convolutions de manière à ce que chaque ligne de la matrice de memristor stocke tout les poids des kernels d'une couche de sortie de la convolution. Cela permet d'utiliser une matrice de memristors bien plus petite, mais il faut alors effectuer plusieurs

produits matrice-vecteur pour obtenir le résultat complet d'une couche de convolution. De plus, plusieurs des sorties seront affectées par le même bruit, ce qui ajoute de la corrélation.

Après avoir défini les modèles de calcul de chaque couche du RNP, notre objectif est de calculer l'EQM entre la sortie du réseau de neurones implémenté sur memristor et la sortie du même réseau implémenté sur un matériel fiable. Nous montrons que cela nécessite de calculer les premiers et seconds moments en sortie de chaque couche du réseau, en propageant donc les moments dans les couches successives. Pour les couches linéaires, il est aisé de calculer les moyennes, variances et covariances. En revanche, pour les fonctions d'activation non linéaires, il est nécessaire d'utiliser des approximations. Pour cela, nous comparons deux méthodes possibles. Dans la première méthode, des dévelopements de Taylor sont utilisés pour approximer les moments d'ordre un et deux en sortie de la fonction d'activation. Dans la deuxième méthode, nous considérons que les entrées suivent une loi gaussienne et ainsi en connaissant la loi de probabilité des entrées, il est possible de calculer directement leurs moyennes et variances à la sortie d'une fonction non linéaire. Nous proposons ainsi des formules analytique dans le cas d'une fonction d'activation ReLU. Nous introduisons aussi des équations pour estimer la consommation d'énergie des matrices de memristors des couches linéaires et de convolutions en fonction de la conductance maximale choisie.

Ainsi, nous avons identifié les conductances maximales de chaque matrice de memristors comme un ensemble de paramètres permettant de réduire la consommation d'énergie de la mémoire et nous avons formulé des équations permettant de mesurer la performance d'un réseau de neurones implémenté sur memristor et sa consommation d'énergie en fonction des valeurs de conductances maximales choisies. Ceci nous permet de formuler un problème d'optimisation où nous cherchons le meilleur ensemble de valeur de conductance maximal permettant de minimiser la consommation d'énergie totale des memristors pour un critère d'EQM fixé. Nous considérons trois cas possible. Dans le premier cas, la même valeur de conductance maximale est fixée pour tout le réseaux. Dans le deuxième cas, une valeur de conductance maximale différente est utilisée pour chaque couche du réseau. Et finalement dans le troisième cas, une valeur de conductance maximale différente est utilisée pour chaque colonne de matrice de memristors. Afin de résoudre ce problème d'optimisation, nous proposons d'utiliser une méthode heuristique d'optimisation en utilisant un algorithme génétique.

Nous présentons des résultats sur deux RNP différents : un premier petit réseaux composés de deux couches pleinement connectées et appliqué à un problème de régression, et un deuxième plus grand réseau convolutif appliqué à une tâche de classification sur le set de données CIFAR-10 [18]. Nos résultats permettent de vérifier que l'EQM estimée par notre analyse est en bonne adéquation avec des simulations, ce qui valide nos équations. De plus, nous montrons

qu'une implémentation efficace de notre analyse théorique permet d'estimer l'EQM du réseau de neurones de 200 à 2000 fois plus rapidement qu'en utilisant des simulations avec la méthode de Monte-Carlo. Finalement, l'étude des solutions des problèmes d'optimisation permet de montrer que celles-ci permettent de réduire la consommation d'énergie du réseau tout en obtenant une performance similaire. Cependant avec cette méthode, les gains d'énergie restent léger. Par exemple, il est possible de réduire de 6% la consommation d'énergie des memristors de notre réseaux pour obtenir une précision de 90% sur CIFAR-10.

**Conclusion et perspectives**

Dans cette thèse, nous avons étudié l'implémentation de deux algorithmes différents sur deux types différents de mémoires peu fiables et à faible consommation d'énergie : premièrement, le filtrage de Kalman utilisant une mémoire à tension réduite, et deuxièmement, des réseaux de neurones implémentés sur des matrices de memristors. Malgré le fait que nous ayons étudié des algorithmes différents pour chaque système, nous avons proposé une méthodologie similaire dans chaque cas. Nous avons d'abord spécifié notre modèle d'erreur en fonction du matériel non-fiable étudié, et de l'implémentation de l'algorithme sur ce matériel. Pour chaque modèle, nous avons constaté qu'il existait un ensemble de paramètres qui peuvent être utilisés pour équilibrer le compromis entre le niveau d'erreur de la mémoire et sa consommation d'énergie. Nous avons ensuite utilisé ces modèles pour estimer l'impact du bruit sur les performances de l'algorithme. Pour cela, nous avons fourni des équations analytiques capables d'estimer l'EQM entre la sortie de l'algorithme implémenté sur du matériel non fiable et une implémentation fiable. Ces équations consistent à propager l'estimation des moments de premier et second ordre de la sortie des étapes de calcul successives de l'algorithme étudié. Nous avons ensuite utilisé ces équations pour estimer la dégradation des performances de l'algorithme considéré et sa consommation énergétique. Cela nous a permis de proposer des méthodes d'optimisation pour calculer les meilleurs paramètres capables de minimiser la consommation énergétique de la mémoire tout en atteignant une certaine performance cible de l'algorithme. Dans le cas des filtres de Kalman, nous avons proposé une solution analytique au problème d'optimisation tandis que pour les RNP, nous avons fourni une solution heuristique. Dans les deux cas, nous avons réalisé des expériences approfondies pour prouver l'exactitude des analyses théoriques proposées par rapport aux simulations de Monte-Carlo. De plus, nous avons montré que la solution à notre problème d'optimisation pouvait conduire à des gains d'énergie, avec un gain allant jusqu'à 50% dans le cas du filtre de Kalman.

Dans cette thèse, nous avons cherché à introduire une méthodologie générique, qui pourra par la suite être adaptée à d'autres modèles d'erreurs et algorithmes. Dans un premier temps,

cette méthodologie pourrait être appliquée par exemple à des variations du filtre de Kalman standard, comme le filtre de Kalman étendu, qui est capable de prendre en compte la non-linéarité des modèles du système. De même, d'autres types de RNP pourraient être étudiés avec une implémentation de memristors, puisque notre méthodologie peut facilement s'étendre à d'autres types de couches.

La méthode proposée pourrait également être améliorée de différentes manières. L'un des principaux axes d'amélioration serait d'utiliser des modèles d'erreur et d'énergie plus réalistes. Dans le premier cas, l'analyse pourrait être enrichie en considérant des hypothèses plus complexes mais aussi plus réalistes, comme un bruit non indépendant et identiquement distribué ou corrélé temporellement ou spatialement. Concernant l'estimation de l'énergie, dans le cas du filtre de Kalman un modèle plus précis pourrait prendre en compte le nombre exact d'accès en lecture et en écriture à la mémoire. En outre, la combinaison de notre méthode théorique avec une conception de circuit fonctionnel réelle qui peut être utilisée pour effectuer des simulations VHDL pourrait aider à obtenir une meilleure compréhension, plus précise, des gains énergétiques qui pourraient être réalisés dans des implémentations pratiques. De même, dans le travail sur les RNP utilisant des memristors nous avons regardé uniquement la consommation d'énergie des matrices de memristors, mais nous avons ignoré les autres composants des circuits tels que les convertisseurs numériques-analogiques à l'entrée des memristors, ou les systèmes numériques effectuant les autres opérations du réseau. Ceci pourrait être réalisé à l'aide de simulateurs d'énergie existants, comme celui présenté dans [19] qui est capable d'estimer la consommation d'énergie d'un RNP implémenté avec des memristor pour une architecture spécifique.

Un autre point d'amélioration réside dans le cas des RNP peu fiables. Le gain d'énergie après optimisation était plus faible qu'espéré. Cela signifie très probablement que le calcul du facteur d'échelle optimal une fois le réseau entraîné n'est pas le meilleur moyen d'obtenir des gains d'énergie significatifs. Cependant, nous pourrions obtenir de meilleurs résultats en calculant les facteurs d'échelle directement pendant la phase d'entraînement, en même temps que les poids. De même, notre méthode d'estimation de l'EQM pourrait être intégrée dans la fonction de perte du réseau pendant l'entraînement. Comme alternative à l'injection de bruit, un terme pour minimiser l'EQM calculé à partir de notre analyse théorique pourrait être ajouté à la fonction de perte afin que le réseau soit entraîné pour minimiser également l'erreur causée par le bruit du memristor. Néanmoins, la mise en œuvre de cette méthode nécessite de programmer des kernels personnalisés capables de rétropropager les gradients à travers notre estimation théorique de l'EQM, et pourrait rapidement se heurter à des problèmes de taille mémoire.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| ANT | Algorithmic Noise Tolerance |
| ASIC | Application-Specific Integrated Circuit |
| CIM | Computing In Memory |
| CMOS | Complementary Metal Oxide Semiconductor |
| CNN | Convolutional Neural Network |
| DNN | Deep Neural Network |
| DRAM | Dynamic Random Access Memories |
| FGPA | Field-Programmable Gate Arrays |
| HRS | High Resistance State |
| KKT | Karush–Kuhn–Tucker |
| LDPC | Low Density Parity Check codes |
| LRS | Low Resistance State |
| ML | Machine Learning |
| MSE | Mean Squared Error |
| MVM | Matrix Vector Multiplication |
| NTC | Near-Threshold Computing |
| NAS | Neural Architecture Search |
| PIM | Processing In Memory |
| SP | Signal Processing |
| SRAM | Static Random-Access Memory |
| TIA | Trans Impedance Amplifier |

# CHAPTER 1 INTRODUCTION

## 1.1 Energy Consumption of Electronic Devices

As new technologies and applications emerge, the energy consumption of electronic systems have considerably risen over the years. The part of the information and communication technology (ICT) sector in the global electricity use passed from 3.9% in 2007 [1] to 7% in 2020 [2]. Further, according to [2], the global energy footprint of communication technology could double by 2030. This significant increase in energy needs of the ICT sector can be explained both by the growing number of electronic systems and by an increase in the energy requirements of digital systems and their applications. As an example, Figure 1.1 shows the number of parameters that need to be stored for different state-of-the-art deep neural networks (DNN) used for image classification since 2012. As new DNN architectures are developed to achieve a higher accuracy, this comes at the price of much larger number of parameters with significant additional energy costs.

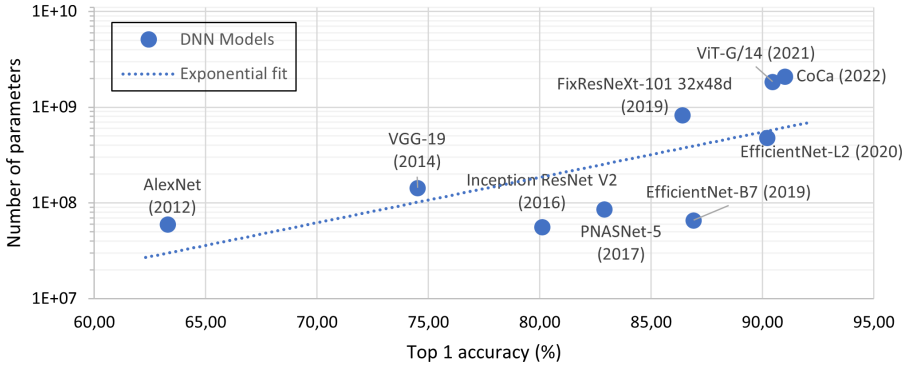



Figure 1.1 Number of parameters (highly correlated with the number of memory accesses) for the top-1 accuracy of state of the art DNN from ImageNet classification competition through the last 10 years [20–27]

At the same time, the number of small embedded devices have also greatly increased. The number of Internet of Things devices was estimated to be 9 billions in 2017 and should reach more than 60 billions by 2025 [3]. Most of these devices will be wireless and they will operate on small batteries or through energy harvesting [4]. This includes biomedical devices [5, 6] and other types of smart sensors [7, 8]. Therefore, there is a rising need for very low power embedded devices capable of running various signal processing (SP) and machine learning (ML) algorithms on a very constrained energy budget.

Since 1965, Moore's law had dictated that the number of transistors on a chip would double every two year, and this increase in transistor density significantly and continuously improved the energy efficient of computer chips until then. However in the last few years, Moore's law based scaling of the energy efficiency of chips components have started to stagnate as shown in [9] for static random access memories (SRAM) and dynamic random access memories (DRAM) memories. Consequently, while until recently, decreasing the size of transistors in complementary metal oxide semiconductor (CMOS) technology led to significant energy savings, this is not the case anymore as we are starting to reach some physical limits.

It is now widely recognized that memories represent the largest part of the energy consumption of electronic devices. Indeed, the memory system is responsible for more than 50% of the total power usage of a CPU chip [10]. For a simple arithmetic operation, memory access can cost between two to three orders of magnitude more energy than the actual operation [11]. Therefore, it is essential to improve the energy efficiency of SP and ML algorithms, and especially of DNNs which are now widely used in various fields. Therefore, in this thesis, we focus on the power usage of the memory in order to reduce the total energy consumption. This should help the emergence of novel applications capable of running SP and ML algorithms on energy-constrained systems with a high level of performance.

## 1.2 Proposed Approach for Lowering the Power Consumption of Memory Systems

Existing methods to reduce the energy consumption of electronic devices can be divided into two main classes. On the one hand, software modifications consist in modifying the algorithms to reduce their power usage. On the other hand, hardware modifications consist of optimizing various design parameters in the hardware implementation such as changes in the component architecture or the use of different component technologies. Finally, the best solution is often an hybrid one, with the energy-efficient design of both software and hardware.

In this thesis, we work on algorithm modifications dedicated to specific emerging hardware solutions to improve the energy efficiency of SP and ML algorithms. We focus on new memory technologies which can have a lower energy consumption than a standard SRAM. We first study the case of a voltage-scaled SRAM with a tunable energy supply. We then investigate in-memory computing architectures based on resistive memories also known as memristors. These two types of memories have seen an increase in interest over the last years due to their potential to decrease the contribution of the memory in the total energy consumption of an electronic system [17, 28]. However, energy gains raised by each technology rely on

different mechanisms. First, a voltage-scaled SRAM works like a standard SRAM, except that the power supply of read-accesses can be adjusted depending on the needs. While resistive memories can significantly reduce the need for transferring data between memories and processing units, by allowing one to implement a part of the computation operations directly in the memory. Yet, they do possess one common trait which is that the energy reduction they allow comes at the cost of introducing errors in their stored values. These errors can be modeled as noise introduced on the computations realized by the device using these unreliable memories.

In this work, we study how errors introduced in the computations by the aforementioned unreliable memories can affect some key SP and ML algorithms. It is important to note that the effect of computation errors onto the performance depends on the considered algorithm. Some of them are inherently robust to errors, while some others can be modified for improved robustness. However, such techniques to improve robustness to errors also consume some energy. Consequently, for a given algorithm implemented on a certain unreliable system, we can define an optimization problem that consists of minimizing the system energy consumption while guaranteeing a certain level of performance despite computation errors. In this work, for several SP and ML algorithms of interest, we aim to identify and solve such optimization problems, by following an approach in three steps:

Step 1 : Provide a theoretical analysis of the influence of the noise caused by the unreliable memory on the algorithm performance. To measure its impact, we use the metric of the mean squared error (MSE) between the output of the algorithm when using an unreliable memory, and its correct version provided by the algorithm implemented on a completely reliable system.

Step 2 : Express both the MSE and the memory's energy consumption as a function of the memory noise level and other parameters of the studied algorithm.

Step 3 : Formulate and solve the optimization problem related to the tradeoff between the algorithm performance and the system energy consumption.

In this thesis, we apply these objectives to two important methods, one from the SP field, and the second from the ML field: we first consider Kalman filtering and then DNNs, both implemented from unreliable hardware.

Although Kalman filtering was introduced more than 60 years ago [12], it is still widely used for a variety of applications, especially for guidance and navigation [30] and biomedical devices [31]. It is often implemented on resource-limited hardware, and as such it is desirable

to improve its energy-efficiency. DNNs on the other hand have seen a growing interest more recently. By playing on their number of parameters, they are known to be useful not only for small energy-constrained applications, but also for larger applications with strong performance objectives. These two algorithms are designed to handle noisy data which makes them prime candidates for also handling perturbations in their computations. Furthermore, both of them are based on iterative calculation, where the input of a computation step uses the output of the previous step. This signifies that in both cases, errors introduced at one step can propagate and accumulate across the next computation steps, making it even more important to compensate for them. Indeed, the Kalman filter consists of successive estimations at successive time instants, while a DNN is built from a large number of successive layers. Therefore, our insight is that such similarities between these two algorithms can lead to similar frameworks for improving their energy-efficiency. However, a standard Kalman Filter is composed of only linear operations, compared to a DNN which also has non-linearities introduced by its activation functions. In what follows, we study the implementation of a Kalman filter onto voltage-scaled SRAMs, while we investigate DNNs from memristor crossbars designed for in-memory computing. This leads to different error models in each case.

### 1.3 Thesis Contributions

In the second chapter of this thesis we review the existing literature on improving the energy consumption of electronic systems and on existing methods for error compensation of SP and ML algorithms using unreliable systems.

Then in the third chapter, we investigate a Kalman filter implemented using an unreliable voltage-scaled SRAM. We consider that it is possible to design the circuits such that the reliability of the memory can be adjusted separately for each bit position. The probability of a bit-flip in a memory cell is a decreasing function of its power supply. We study theoretically how the errors from the memory propagate into the successive estimations of the filter and how this affects the error on the estimation results. We then use this theoretical analysis to formulate new analytical equations of the Kalman filter capable of taking into account this new source of noise as well as quantization noise. We then use these analytical results to formulate an optimization problem for balancing the trade-off between performance of the Kalman filter and energy consumption of the memory. Finally, we propose an algorithm to solve this optimization problem and simulation results show that it can lead to a 2.5× reduction in energy consumption.

Finally in the fourth chapter of this thesis, we study DNNs implemented using unreliable memristors. We formulate theoretical equations capable of predicting the MSE of the output of a memristor-based DNN comprised of a variety of layer types, depending on the memristor noise. From an efficient software implementation of this framework we show that it can be 2 to 3 orders of magnitude faster than Monte Carlo simulations for predicting the performance of the network. We then use the proposed analytical framework to formulate an optimization problem for preserving the accuracy of the network while minimizing the energy consumption of the memristors. Finally we present results on two types of DNNs, including a convolutional neural network capable of achieving a high accuracy on the CIFAR-10 dataset. These results show how accurately our analysis matches the simulations.

## 1.4 List of Publications

The findings of this thesis have been shared through the following publications in journals and international conferences.

### Journal Papers

- J. Kern, E. Dupraz, A. Aïssa-El-Bey, L. R. Varshney, and F. Leduc-Primeau, "Optimizing the Energy Efficiency of Unreliable Memories for Quantized Kalman Filtering," Sensors, vol. 22, no. 3, p. 853, Jan. 2022, doi: 10.3390/s22030853.
- J. Kern et al., "Fast and Accurate Output Error Estimation for Memristor-Based Deep Neural Networks," IEEE Transactions on Signal Processing (Submitted 02/02/203)

### Conference Papers

- J. Kern, E. Dupraz, A. Aïssa-El-Bey and F. Leduc-Primeau, "Improving the Energy-Efficiency of a Kalman Filter Using Unreliable Memories," ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 2021, pp. 5345-5349, doi: 10.1109/ICASSP39728.2021.9413430.
- J. Kern et al., "MemSE: Fast MSE Prediction for Noisy Memristor-Based DNN Accelerators," 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Incheon, Korea, Republic of, 2022, pp. 62-65, doi: 10.1109/AICAS54282.2022.9869978.

# CHAPTER 2 LITERATURE REVIEW

In this Chapter, we describe existing classes of solutions that allow to address the challenge of reducing the energy consumption of a system while preserving its performance. From a hardware design perspective, several possibilities have emerged such as near threshold computing (NTC) or computing in memory (CIM) which we will detail further in this chapter. Additionally, it is also possible to further reduce the energy consumption of a memory system by reducing the number of bits. As the number of possible values for the quantization decreases, so does the number of memory cell needed to store the algorithm data and therefore so does the total energy consumption of the memory.

All the previous methods have the drawbacks of introducing errors in the systems, making them unreliable. However, some algorithms inherently tolerate some level of errors, especially the ones which are already used to processing noisy inputs. To further improve the robustness of these algorithms, and for non-robust ones, there is a need to add mechanisms to compensate for the unreliability. Some of these methods are dependant on the algorithms, whereas others are more generic and can be put upstream or downstream of different types of algorithm.

In this chapter we will first study existing ways of reducing the energy consumption of a system. We will then look into how algorithms can tolerate unreliability and the existing means to compensate for them. In most of the literature, the error compensation methods proposed are associated to a particular algorithm and implementation. We therefore structure this literature review in the same way and present different works with their specific algorithm and system and associated error compensation mechanism.

## 2.1 Hardware Solutions for Reducing Energy Consumption of a System

### 2.1.1 Near Threshold Computing

An introduction to near threshold computing (NTC) can be found in [28,32]. NTC consists in reducing the input voltage of a CMOS circuit towards the threshold voltage of the transistors. This provides two benefits: first, the energy consumption of the system is reduced, second, the energy leakage of the system are reduced exponentially which improves even more the energy efficiency. At the threshold voltage, while it is still possible to use the circuit for computations, its operating frequency is greatly reduced and delays start to introduce obstacles. Especially, the circuit performance is lowered, and shows a lot more variabilities. A certain amount of errors such as bit flips can happen, as can be seen in Figure 2.1 for a 90nm SRAM and a

65nm SRAM. In addition, the loss in performance in term of delay has been shown to be

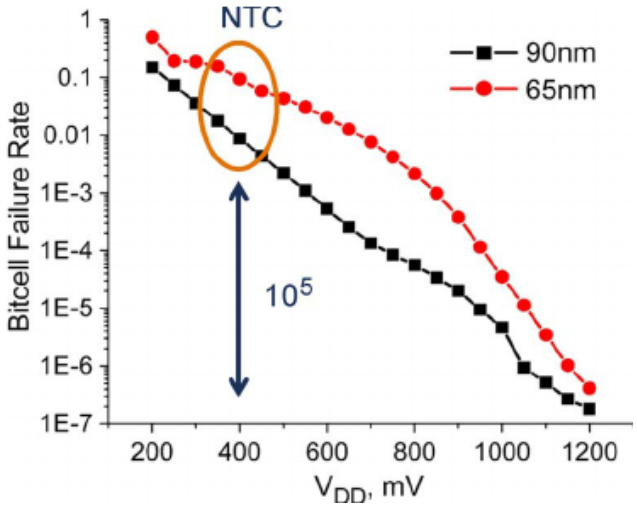



Figure 2.1 Impact of voltage scaling on SRAM failure rates © 2010 IEEE [28]

exponentially proportional to the decrease in energy supply [33]. A 90% reduction in energy, which corresponds to the minimum energy threshold allowing the system to function, can create a drop of 1000 times the performance. On the other hand, an energy increase of only 20% more than the minimum energy point will increase the performance by an order of magnitude.

In their work, [34] studies in more details nanoscale semiconductor devices whose reliability depends on energy consumed. It introduces some exponential functions to link the probability of failure of a logic gate to its energy supply, where a gate failing means that its output bit is flipped. The same type of relationship is found in [13], which presents a system-on-chip (SoC) using an SRAM which voltage can be scaled. The experiments of [13] show that the Bit Error Rate of the memory is indeed exponentially proportional to the inverse of its supply voltage.

### 2.1.2 Computing in Memory

In a conventional computing architecture, commonly referred to as a von Neumann architecture, data is transferred back and forth from the memory to the processing unit. With the paradigm known as Computing in Memory it is possible to reduce the need for transferring the data stored in the memory to the processing unit and back as the computations can be done directly in the memory. Figure 2.2 illustrates the differences between a conventionnal computing architecture and a CIM architecture. In [17] the types of devices used for creating

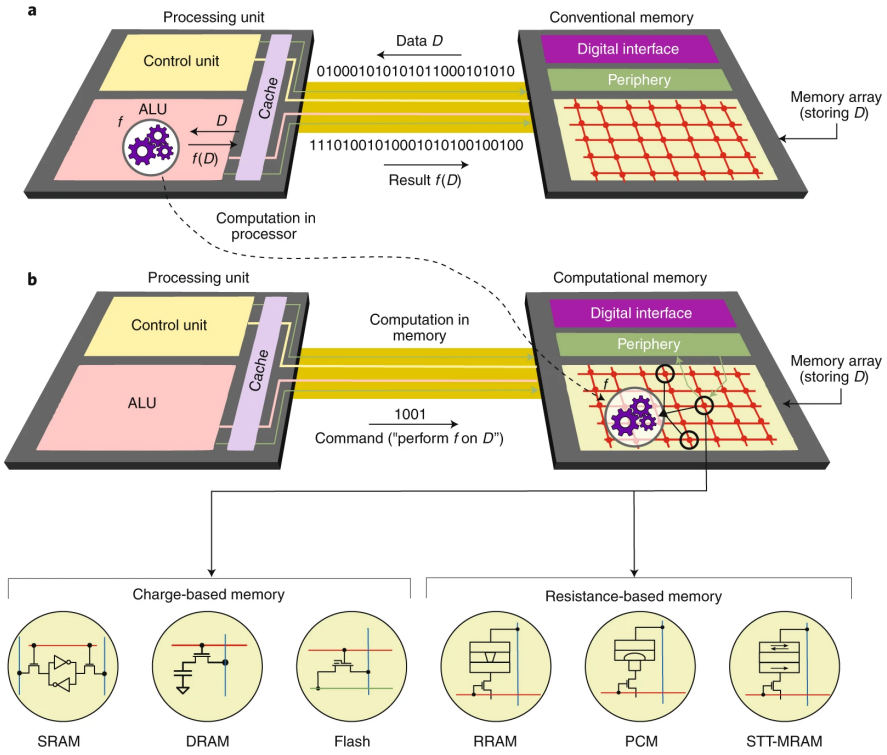



Figure 2.2 Comparision between (a) a conventional computing architecture following a Von Neuman paradigm and (b) a Computing-in-Memory design [17]. Reproduced with permission from Springer Nature

a CIM architecture are separated in two categories: charged-based memory and resistance-based memory. Charged-based devices includes the conventional SRAM and DRAM which also suffer from the scaling problems described above. The memory needs of emerging applications will make them require either a very large area or increase the impact of thermal noise and capacitor size variation. These issues may lead to a decrease in the maximal accuracy achievable by these types of memories. On the other hand, resistance-based memory devices can provide a highly increased data density compared to charged-based memories. Resistance-based memories such as memristors [16] are still an emerging technology, and a lot of work still remains to make them cost-effective and reliable. Indeed, unreliability due to conductance variation represents one of the one of the biggest issues with memristive devices.

### Memristors

Memristors were first theoretically proposed in 1971 by Leon Chua [16]. A memristor is a passive component capable of linking electric charge and magnetic flux. Using memristor crossbars as non-volatile memories present several advantages. First, compared to a con-

ventional SRAM, they can have a far better density of information [35]. Memristors have a resistance value ranging between a low resistance state (LRS) and a high resistance state (HRS). By applying a series of specific voltage, it is possible to set the conductance value of the memristor to a specific target. Therefore, they also allow to realize some computation operations in memory such as matrix vector multiplication (MVM) [36]. In this case, the vector components are applied input voltages to the rows of the memristor crossbar where they are multiplied at each node of the crossbar with the conductance value stored by the memristor. Thus, the output current vector at the end of each column is the result of the multiplication between the input voltage vector and the conductance matrix stored in the memristor crossbar.

### Memristors Variabilities

However, due to several physical issues, as well as device-to-device variations and cycle-to-cycle variations, the conductance value stored in a memristor can be different from the desired value and noises or errors can be introduced in the computations, making them unreliable. Another source of unreliability comes from "sneak paths" where the current traversing a memristor crossbar can pass through undesired paths [37]. The fabrication process also introduces device-to-device variations or cycle-to-cycle variations [38]. The latter ones are introduced by irregularities in the shape of the conductive filament of the memristor and by other microscopic variabilities around the filaments [39].

Various statistical models have been proposed to take these variabilities into account. For example in [40], which studies Hamming distance computations from memristor crossbars, the resistance variations are modeled by a Gaussian noise, and the write errors between the HRS and the LRS due to the switching mechanism are modeled as a binary symmetric channel. The measurements realized in [29] also exhibit a Gaussian distribution of the conductance values of the HRS and LRS, where the variance of the distribution is shown to depend on the considered programming scheme used. The results of [29] can be seen in Figure 2.3.

To circumvent some of the issues caused by memristors, composite cells associating another element to the memristor have been developed such as one-transistor-one-memristor cell (1T1R) [41] or one-dione-one-memristor (1D1R) [42]. Associating a transistor to each memristor is bound to greatly increase the area need for a crossbar compared to having only 1R cells. However, the transistor allows to more efficiently program the conductance value to a memristor by avoiding sneak-path currents [43].

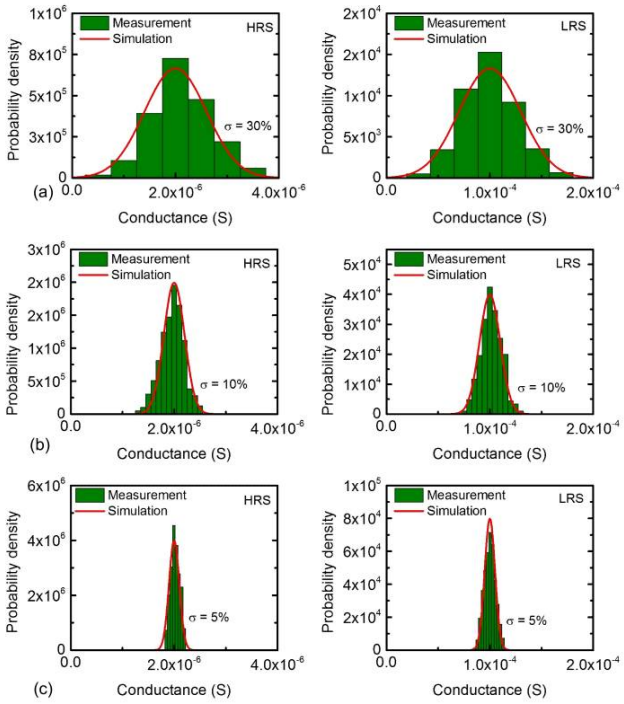



Figure 2.3 Memristance distributions of HRS and LRS for different programming schemes [29] - Licensed under CC BY 4.0

### Memristors in Machine Learning Applications

In the past few years, a wide range of works studied the use of memristors in machine learning. For instance, [44] states that memristors crossbars can process MVM with a O(1) complexity, which makes them particularly useful for neural networks. As such, [45] proposes a 1R methods for implementing a neural network where one memristor is used to represent each weight in the network. In this case, each layer in the network is represented by a memristor crossbar. In [46], 1T1R cells are used. Having a transistor associated to each memristor cell helps improve the programming accuracy by controlling and limiting the current. 1T1R cells are also used in [47] but in their case two memristors cells are used to represent one DNN weight which allows them to double the precision of the weight stored in their memory. This design also has the second advantage that the weights are on average stored with a lower conductance state compared to the one memristor per synapse case. Due to the lower conductance values, the energy consumption of the memristors is 9 times lower than the single memristor design. The same design is used in [48].

### Quantized Memristors

Although memristors can theoretically store any value since their conductance is analog, it is often useful to store quantized values for robustness purposes. In this way, [49], [29] and [50] store binary values to lower variability issues and better distinguish between memristor states: the greater the ratio between the LRS and the HRS, the better the reliability of the memristor circuit. Similarly, [51] studies quantized memristive neural networks with different levels of quantization. They propose to group binary memristors to represent one weight with more precision while decreasing their influence to variations. They show that using $n$ binary memristors to represent $n$-bit weights allows for an increased robustness compared to using $n$-level memristors. For binary memristors, the smaller the precision, the smaller the area of the memory circuit and the faster the computation time, but using one multi-level memristor per synapse allows to have an even greater area density and energy efficiency.

### Writing Values to Memristors

If the training of the neural network is done on memristor crossbars, then this process can be very consuming in energy due to the power usage of programming memristor values. To program the memristor values, a "program-verify" scheme is used where series of alternating programming and verify pulses are used until the correct value is read from the memristor. By reducing the number of pulses, it is possible to decrease the energy consumption of the programming scheme but on the other hand, this reduction will have the effect of increasing the variance of the programmed conductance Gaussian distribution. In [29], they show that to obtain the best trade-off between energy consumption and performance of the network it is best to use more energy for the programming of LRS values and less energy for HRS values. A similar method is used in [52] where depending on the programming current, the gap between the distribution of the LRS and HRS will change. The lower the programming energy, the more the distribution between the two states will overlap and therefore, the more the bit error rate increases. Due to neural networks tolerance to errors, [52] demonstrates that it is possible to decrease the programming energy of the memristors while still preserving the performance of a CNN despite the bit errors.

### Processing Different Types of NN Operations on Memristor Crossbars

Memristors are particularly adapted for storing the weights of a neural network and computing MVM but their use is more complex for doing the other types of computations required by a neural network. This is why they are often associated with a digital CMOS circuit. To

circumvent this constraint, [53] presents a full-analog implementation of back-propagation for neural networks training. This would allow to avoid the need for analog-digital and digital-analog converter at the input and output of the memristor circuits. However, more investigation needs to be done on the influence of the memristor variabilities on these computations as the training process of a neural network is less tolerant to errors. Further work has been done for implementing specific types of neural networks which require other types of computations on memristor systems such as recurrent networks with LSTM [54] or auto-encoders [55].

#### Memristors Simulation Frameworks

Several frameworks have been developped in the past few years to allow the simulation of crossbar arrays for DNN computations [56, 57]. Theses frameworks are capable of simulating the errors and variabilities of memristor computations. In [56], the proposed framework is used to simulate the implementation of several DNN on the ImageNet classification task [58]. The simulations reveal that errors introduced by the memristors can degrade the classification accuracy by up to 32%. This leads the authors to conclude on the importance of adding mechanism for error compensation or correction.

### 2.2 Algorithmic Solutions for Reducing the Energy Consumption of a System

#### 2.2.1 Quantization

Quantization has long been used in the field of electronic systems to reduce energy consumption and improve the processing efficiency. The design objective is to minimize the error introduced by the quantization process while still achieving significant energy savings. This has been used in many applications such as MMSE precoder used for massive MIMO with 1 bit quantization [59], reconstruction at the fusion center and distributed estimation in wireless sensor networks [60, 61], transmission of sensing data in battery-limited medical devices [62] or for applications using discrete cosine transform [63].

Quantization is also widely used in neural networks [64, 65]. Reducing the number of bits for storing the weights or the activations leads to computations with reduced precision, but on the other hand this reduces the memory and power requirements. One of the way to compensate for the loss of precision can be to modify the architecture of the neural network. This can be formulated as addressing the trade-off between quantization precision and network size, so as to minimize the energy consumption while maintaining the best possible performance. Depending on the quantization level, [66] found that the difference between

energy requirement of different network architectures can be more than 20 times. Other recent advances have even shown that it is possible to aggressively scale the quantization of a neural network to 4-bits while still preserving complete accuracy of the network [67].

### 2.2.2 Approximate Computing and Error-Resilient Algorithms

#### Approximate Computing

The concept of approximate computing [68] follows the same line as NTC. Thanks to their resilience to errors, algorithms that use this paradigm are able to achieve results equal or close to their maximum performance despite a loss in exactness of their computations. For instance, estimation algorithms already process noisy signals, which often makes them robust to other sources of errors up to a certain type and amount. The category of error tolerant algorithms also cover many error-correction decoders since they are used to detect and correct errors in their inputs. Different decoding algorithms have already been investigated under unreliable conditions such as min-sum decoders [69] and bit-flip decoders [70] for LDPC codes, or turbo code decoders [71].

Incremental refinement [68] is a first approximate computing technique for iterative algorithms for which more iterations lead to a more precise solution. It consists of terminating the algorithm earlier to obtain some energy savings at the cost of a less accurate result. This idea was used for example in a Fast Fourier Transform-based maximum-likelihood detection algorithm [72].

In [73], a framework for characterizing the inherent resilience of signal and data processing algorithms is proposed. The analysis of [73] shows that for an algorithm to be robust to errors, the faults can either be frequent but small, or rare but of a possibly high magnitude. Algorithms such as K-means clustering [74] and SVM [75] are proved to be error resilient.

#### Error Tolerance of Neural Networks

Neural networks are also prime candidate for approximate computing as they have shown a tolerance to errors by several aspects. For example, [76] experimented with a DNN implemented using a voltage-scaled SRAM and showed that under a certain threshold of noise one can achieve more than 99% of the reliable floating point test accuracy. Furthermore, [77] showed that if the accuracy of the network is degraded due to faults in the computations, the loss in accuracy can be compensated by increasing the size of the network. This leads to a similar problem to quantization as we have a trade-off between the size of the network and its energy supply to minimize the total energy consumption. It should be noted that

depending on the magnitude of the unreliability of the system, it is not always possible to reach the equivalent performance of a reliable network.

DNN implemented using unreliable voltage-scaled SRAM are also studied in [78] which shows that it is sometimes more energy efficient to reduce the supply voltage rather than decreasing the number of parameters. However, it is stated that there is still work to do for finding the best network architecture capable of showing the best error tolerance and the best accuracy simultaneously. In [78], it is also noted that once the bit error rate of the memory becomes too large, a sharp loss in performance can happen. For neural networks, [73] also observed that the larger the size of the training data was, the more the trained algorithm would be resilient to errors.

Finally, [79] looked into the error tolerance of Binarized DNNs and showed that using binary weights can soften the impact of errors as no bit is more significant than another. On the other hand, as DNNs are composed of a series of layers, they are subject to error accumulation. They observed that the total error level increases after each linear layer, while the pooling layers which subsample the output from previous layers reduce error accumulation, thus preserving in part the accuracy. Additionally, their experiments demonstrated that bit errors have more effect on the weights than on the activations.

## 2.3 Systems Improved by Noise and Stochastic Resonance

Stochastic resonance refers to algorithms that can actually be improved by specific types of noise. This phenomenon was observed for parameters estimation [80], image processing [81], or to shorten the convergence time to equilibrium of finite-state Markov chains [82]. The concept of noise-enhanced systems can also be found in the case of neural networks. For instance, [83] shows that injecting a specific noise can improve the speed of convergence in the training of a convolutional neural network. However, as mentioned in [80–83], stochastic resonance is limited to certain algorithms and is only for specific types of noise.

## 2.4 Existing Methods of Error Correction and Compensation for Unreliable Systems

### 2.4.1 Algorithmic Modification for Compensating the Loss of Robustness

#### Algorithmic Noise Tolerance

Algorithmic noise tolerance (ANT) [84, 85] is a general method for compensating the noise introduced by voltage-scaled hardware. With ANT, a system implementing an algorithm is

decomposed into two blocks. The first main block processes the algorithm on the voltage scaled system. To compensate for the unreliability, a second computing block is introduced for estimating and limiting the error of the main block. The interest of ANT was demonstrated on many applications such as filtering algorithms [85] and Viterbi decoders [86]. There exists different ways to implement the ANT block. In [84], the potentially erroneous result out of the main block is supplied as an input to an error-control block. If an error is detected, the output is replaced by an estimation of the correct result. This method is applied to frequency selective filtering algorithms and depending on the filter, the authors were able to show energy saving of up to 80% compared to a conventional system for a degradation of the result of around 0.5dB. The challenge with this type of method is to be able to conceive a secondary block capable of detecting and compensating error while consumming as little energy as possible. Therefore, ANT might not be applicable to all systems.

An other example of ANT is given in [87]. In this case, the secondary block performs the same computation as the main block but with a reduced precision. Because of the delays introduced by the voltage scaling of the system, the errors tend to happen on the bits computed last, which are also the most significant bits. So by reducing the quantization precision, the secondary block should not suffer from the same delay errors. Therefore, if an error is introduced in the computation of the main block, the output will vastly differ from the correct result which makes it easy to detect errors in the computation. In that case, the erroneous result is replaced by the approximation of the low precision block. The number of bits for the precision of the second block dictates the trade-off between energy consumption of the system and precision of the approximation. To reduce even further the energy consumption, [88] proposes to integrate the secondary estimation block into the main block, so that its additional computational complexity is reduced, as well as its surface area.

### Error Correction Codes

An other error-correction method consists in encoding the inputs or computations of the algorithm and relying on error correction codes for correcting the errors introduced in the computations. A first example can be found in [89] which uses Bose-Chaudhuri-Hocquenghem coding for designing error-tolerant linear finite-state machines (LSFM). Note that [89] assumes that encoding and decoding are reliable. Similarly, [90] considers low density parity check codes (LDPC) to ensure that a number of identical LFSM running in parallel but with different inputs can operate without errors. Additionally, in [90], the error correcting scheme is done on unreliable hardware. Finally, [91] considers using LDPC codes for performing reliable binary linear transformations using circuits built entirely out of unreliable

components, meaning that the circuits for encoding the data and correcting the errors are also noisy. The noisy decoders are embedded at each step of the computations to avoid the accumulation of errors. Furthermore, in [91], the authors investigate the case of "tunable" supply voltages for the faulty gates used for computations. They show that by dynamically scaling the gates supply voltages, it is possible to reduce the energy consumption by orders of magnitude compared to having a constant supply voltage.

#### Tuning the Energy Allocation

In [92], binary recursive estimation implemented on unreliable hardware is investigated. This paper studies the effect of errors introduced in the computations, and the error probability depends on the system power supply, and develops a theoretical analysis of the performance loss due to errors, depending on the power supply. Using this analysis, the paper formulates an optimization problem for computing the best energy allocation accross time and accross the quantization bits.

### 2.4.2 Error-Correction and Compensation Methods Proposed for Neural Networks

#### Algorithmic Noise Tolerance

First, [93] proposed to use ANT through a method called "Rank Decomposed-Statistical Error Compensation" and dedicated to computing MVM, which are widely used in CNNs. As in other ANT techniques, the computations are made using one main block and one estimator block. The main block performs conventional MVM but with a reduced power supply, making the computations unreliable. The estimator block uses the important redundancy in the multiple MVMs performed in a convolution layer to compensate the errors of the main block while using few resources. For CNNs, the overhead computing cost due to the additional estimator block is between 5 to 15%. Therefore, ANT is worth it if the reduction in energy of the main block allows to reduce the system total energy consumption by more than this overhead. The authors estimate that their method provide a 11× increase in tolerance to variations, compared to a standard CNN. An other technique of Statistical Error Compensation for reducing the energy consumption of a CNN is presented in [94]. The main block is a voltage-scaled implementation of the neural network and the estimator block approximates the MVM by using K-means clustering.

### Injecting Errors During Training

The works mentioned previously focused on studying and mitigating the influence of errors on a trained network. However, noise tolerance can also be addressed during the training phase of a neural network [78]. In [78], a regularization technique is proposed, that consists of applying errors during the forward pass of the training process. Results on the CIFAR-10 dataset [18] show that using this regularizer during training can lead to 2.3 $\times$ reduction in energy consumption compared to using a reliable system for achieving the same accuracy. A similar method called "Memory Adaptive Training" is proposed in [95]. This methods first builds a profile of the errors happening on a voltage-scaled SRAM, and then uses this error profile to inject errors on the weights prior to the forward pass. Therefore, backpropagation will be able to take these errors into account. Error injection is also used for memristive devices in [96]. In this method, a network pretrained with reliable weights is retrained while adding random Gaussian noise on the weights at each forward pass. With this training method, experiments showed a degradation of only 0.5% compared to the baseline accuracy on a ResNet-32 network on CIFAR-10 despite having a noise on the weight with a standard deviation of 5% . In [97], noise injection is also used but with a focus on accurately modeling the noise of the specific target hardware for implementing the neural network. With Hardware-Aware training, [97] models all the non-idealities of an analog CIM architecture so that they are accounted for during the training of a neural network. Using this framework, it is possible to train various types of DNN so that they maintain they baseline accuracy while running on the unreliable hardware they were trained for.

### Computing the Optimal Energy Allocation During Training

In [98], the training process of the DNN is modified to train not only the weights but also the energy supply of the memory for each layer. The weights of each layer are stored on a memory with its own energy supply, which leads to different error rates across layers. Depending on the target trade-off between accuracy and energy consumption, the training process will reduce the energy supply of layers which have less influence on the final result while increasing the energy supply of the most impactful layers. This method is shown to reduce by a factor 3 the energy consumption of the memory for the same level of accuracy. The idea of having different power supply for different parts of the memory was also investigated in [95]. This work shows that the convolutional layers are more sensitive to errors than the fully connected layers. Therefore, the authors propose to use two memory banks with two different supply voltages. The weights of the convolutional layers are stored in the memory with the highest energy supply to be better protected against errors.

### Training to Maximize Error Tolerance

The training methods described above all rely on error models developed for the target hardware. However, new emerging methods propose to train neural network for error tolerance without injecting errors in the training process. The work in [99] proposes to modify the error function used for training the neural network so as to improve its resilience to errors, by adding three regularizing terms to the error function. The first regularizer consists in taking in the norm of the weight vectors, which is shown to enhance the resilience of the network. Moreover, this term shrinks the magnitude of the weights, which speeds up gradient-based learning. The other two terms are the first and second order derivatives of the MSE with respect to the weights. Together, these regularizers allow to avoid that a small adjustment could cause a large change in output error. Thanks to these two terms, the network is less affected by changes in its weights and more robust against errors. The idea of improving the error tolerance of a binary neural network without error injection is also studied in [100]. The authors propose using margin maximization. This consists in maximizing the margin between the first highest and second highest output at the output layer such that even if the output value of the correct class suffers from errors, it will still be the highest value. This margin is incorporated in a modified version of the hinge-loss function so as to work in the case of a multi-class problem. This modified loss function is able to achieve better performance than a network trained with a standard loss function and bit errors injections. However, combining error injection with the new loss function produced even better results. An other method proposed by [101] is sharpness aware minimization, which consists in trying to smooth the loss landscape of the training. Sharpness aware minimization was already known to improve the generalization capacity of a DNN [102], and in [101] it is shown to also improve the robustness against noise. Furthermore, [101] puts in evidence a correlation between the sharpness of the loss and the performance degradation of the trained network.

### Hardware-aware Neural Architecture Search

To try to balance the performance-energy trade-off of a neural network implemented using unreliable hardware, it can also be possible to try and find the best possible DNN architecture that maximize its robustness to errors while minimizing its energy consumption. To accelerate this process, neural-architecture search (NAS) algorithm can automate the search for a DNN design capable of satisfying various specified constraints [103]. More specifically, with hardware-aware NAS it is possible to define the specificity of the target hardware unto which a DNN needs to be implemented and find the optimal architecture that will minimize the energy consumption of this hardware while satisfying a desired accuracy [104, 105].

## 2.5 Conclusion

In this chapter, we provided an overview of different hardware, software, and hybrid solutions for reducing the energy consumption of SP and ML algorithms. Although certain algorithms are inherently tolerant to noise, we have seen that most often there is a need to add an error compensation mechanisms to algorithms implemented on unreliable hardware. Using these techniques, it is possible to decrease the energy consumption of a system while minimizing its performance degradation.

In this thesis, we build on existing works to improve the error tolerance of SP and ML algorithms implemented on unreliable systems. However, rather than adding additional error-compensation mechanisms, we aim to take a different approach, similar to the one used in [92]. Using the error models found in the literature for different unreliable energy-efficient memories, we propose a more analytical approach where we first provide a theoretical analysis of the effect of the noise on the algorithm. We then use this analysis to compute the optimal values of a parameter capable of controlling the trade-off between performance of the algorithm and energy supply. This parameter can be the energy supply of a voltage-scaled SRAM as in [98], or the scaling factor of memristor conductance values as in [106].

# CHAPTER 3 QUANTIZED KALMAN FILTERING IMPLEMENTED ON VOLTAGE-SCALED MEMORIES

## 3.1 Introduction

Kalman filtering is a very common recursive estimation task in statistical signal processing [12], and it is often implemented on resource-limited hardware. Applications that require an embedded energy-efficient Kalman filter include air quality monitoring [107], biomedical wearable sensors [31], forest fire detection [108] and vehicle positioning [30]. Therefore, in the work of this chapter, we focus on optimizing the energy used by memories in Kalman filters. For this, we consider a Kalman filter implemented on a voltage-scaled memory, as described in section 2.1.1.

Although Kalman filtering has not previously been investigated under unreliable hardware implementation, some related works considered this filter and other similar models for linear systems under uncertain conditions. These include errors on the filter's gain [109, 110], sensors failures, uncertainties on the observations [111, 112], or inaccuracies in the filter parameters [113–115]. In these works, new filter equations were derived using the Riccati equations approach to find new bounds or guarantee on the performance of the filter.

While these models are not relevant for characterizing the effect of unreliable memories, the main lessons they provide are that Kalman filtering is very sensitive to inaccuracies and that one should re-derive the optimal Kalman filter depending on the specifically considered uncertainty model. On a different line of research, other prior works aim at reducing the energy requirements for Kalman filtering by focusing on reduced computational complexity in field-programmable gate arrays (FGPAs) [116, 117] and application-specific integrated circuits (ASICs) [118].

To design a digital hardware implementation, variables and computational operations must be quantized. As mentioned in Chapter 2, to further reduce the memory energy consumption, one option is to properly optimize the quantization to reduce the memory requirements of the implementation. The effects of quantization on the Kalman filter were first studied in [119, 120] to understand the convergence of filters with reduced precision. More recently, refs. [121–123] considered two distributed quantized Kalman filters, based on quantized observations and quantized innovations, where sensors process and transmit quantized observations and innovations to a fusion center. Furthermore, [121] proposed to optimize the number of quantization bits at each sensor to minimize the required data transmission energy.

More general linear stochastic systems were also investigated under quantized measurements [124] and quantized innovations [125], where it was shown that the derived quantized filters converged to standard Kalman filters as the number of quantization levels increased. However, none of these theoretical works considered quantized parameters (e.g., quantized Kalman gain matrices, quantized measurement matrices, etc.), in addition to quantized observations/innovations. Therefore, in this work, we study a fully-quantized Kalman filter and investigate its energy consumption when using unreliable memories.

Here, we aim to optimize the energy consumption of a Kalman filter implemented with fixed-point quantization [14] and with unreliable memories. Fixed-point representations are often preferred in energy-constrained systems as a fixed-point operation can consume 10-times less energy than a floating-point one [11]. We consider the statistical model of [28], which relates the amount of faults introduced in memory for its energy consumption. Then, as a first contribution, we propose a unified framework to analyze the performance of Kalman filters with both quantization errors and faults introduced in the memory.

To develop this framework, we build on the approach of [120], which consists of evaluating the covariance matrix of the estimation error at each filter iteration by considering both error propagation from previous iterations and errors introduced at the current iteration. Our analysis also includes quantized filter parameters and further incorporates the effect of unreliable memories. Determining the covariance matrix of the estimation error has two advantages. First, it allows us to derive the optimal Kalman filter equations under the considered quantization and memory error models. Second, and more specific to our case, it defines a performance criterion that will be used to optimize the memory energy consumption.

As a second contribution, we define two optimization problems to minimize the memory energy consumption while satisfying a target constraint on the estimation performance of the Kalman filter. In the first problem, we optimize the number $B$ of quantization bits and the energy allocated to each bit position to minimize the overall energy consumption of the memory. This optimization problem extends to [15], which was not dedicated to Kalman filtering but derived optimal bitwise energy allocations with a fixed number of quantization bits by considering a generic MSE performance criterion when reading a word in memory.

Although a useful baseline, the setting where each bit position can have a different energy allocation is not practical, since each of the $B$ bits should be placed in a different memory bank with its own power supply. This is why we also introduce a second optimization problem in which we fix the number $L < B$ of possible energy levels and optimize the energy value in each level and the mapping of bit positions to an energy level.

At the price of a small energy increase, this optimization problem allows us to build a practical implementation that only requires $L$ memory banks. By using the Karush–Kuhn–Tucker (KKT) conditions, we provide solutions for the two considered optimization problems. Both solutions can be numerically computed using water-filling. Numerical simulations show that, after optimization, the memory energy consumption is reduced by up to 56% compared to uniform allocation.

The rest of the chapter is organized as follows. Section 3.2 describes the quantized Kalman filter and introduces the uncertainty model for unreliable memories. Section 3.3 investigates the theoretical performance of the filter. Section 3.4 formally defines and solves the two considered optimization problems. Section 3.5 presents the simulation results.

## 3.2 System Model

We first review the Kalman filter for estimating dynamic state variables from noisy measurements. We then present the considered implementation of the filter by first introducing its quantization model and then describing its implementation with an unreliable memory.

### 3.2.1 Kalman Filter

The process:

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{u}_k \,, \tag{3.1}$$

describes the linear dynamic variable $\boldsymbol{x} \in \mathbb{R}^c$, where the state vector of the process at step $k$ is noted as $\boldsymbol{x}_k$, $\boldsymbol{F}$ is the state transition matrix of size $c \times c$ , and $\boldsymbol{u}_k \in \mathbb{R}^c$ is an additive white noise vector [126]. Observation of the state of $\boldsymbol{x}$ can be obtained through $\boldsymbol{y} \in \mathbb{R}^d$, the measurement vector defined as

$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{v}_k \,. \tag{3.2}$$

Here, $\boldsymbol{H}$ is the $d \times c$ measurement model and $\boldsymbol{v}_k \in \mathbb{R}^d$ is an additive white noise on the measurements, independent from the model noise $\boldsymbol{u}_k$. We denote $\boldsymbol{Q}$ and $\boldsymbol{R}$ as the known covariance matrices of the noise vectors $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$, respectively.

Using the knowledge of the model as well as the the measurement vectors $\boldsymbol{y}_k$, the Kalman filter [12] recursively estimates the successive states $\boldsymbol{x}_k$. This is done by minimizing the mean squared error between $\boldsymbol{x}_k$ and its estimate $\hat{\boldsymbol{x}}_k$ at each step $k$: $\mathrm{MSE}(\boldsymbol{x}) = \mathbb{E}[\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k\|^2]$. The filter can be decomposed in two phases: the a priori estimation uses only the known model, and the a posteriori estimation takes into account the measurements.

At each phase, both the estimates $\hat{\boldsymbol{x}}_{k+1|k}$ (for the a priori phase) and $\hat{\boldsymbol{x}}_{k+1|k+1}$ (for the a posteriori phase) of the state vector $\boldsymbol{x}_{k+1}$, and the covariance matrices of the estimations errors $\boldsymbol{P}_{k+1|k} = \mathrm{Cov}[\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}_{k+1|k}]$ and $\boldsymbol{P}_{k+1|k+1} = \mathrm{Cov}[\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}_{k+1|k+1}]$ are computed. The recursive equations of the a priori estimation step are [127]:

$$\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{F}\hat{\boldsymbol{x}}_{k|k}\,, \tag{3.3}$$

$$\boldsymbol{P}_{k+1|k} = \boldsymbol{F}\boldsymbol{P}_{k|k}\boldsymbol{F}^{\top} + \boldsymbol{Q}\,, \tag{3.4}$$

and the recursive equations of the a posteriori estimation step are:

$$\boldsymbol{K}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{H}^{\top}(\boldsymbol{H}\boldsymbol{P}_{k+1|k}\boldsymbol{H}^{\top} + \boldsymbol{R})^{-1}\,, \tag{3.5}$$

$$\hat{\boldsymbol{x}}_{k+1|k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{K}_{k+1}(\boldsymbol{y}_{k+1} - \boldsymbol{H}\hat{\boldsymbol{x}}_{k+1|k})\,, \tag{3.6}$$

$$\boldsymbol{P}_{k+1|k+1} = (\boldsymbol{I} - \boldsymbol{K}_{k+1}\boldsymbol{H})\boldsymbol{P}_{k+1|k}\,, \tag{3.7}$$

where $\boldsymbol{A}^{\top}$ denotes the transpose of a matrix $\boldsymbol{A}$. In these equations, the Kalman gain $\boldsymbol{K}$ of size $c \times d$ and the covariance matrices $\boldsymbol{P}$ of size $c \times c$ can be computed offline. On the other hand, the terms $\hat{\boldsymbol{x}}_{k+1|k}$ and $\hat{\boldsymbol{x}}_{k+1|k+1}$ must be computed online as they depend on the measurements $\boldsymbol{y}_k$.

### 3.2.2 Quantized Implementation of the Filter

In the rest of the work, we study Kalman filters that are implemented under fixed-point quantization [14]. Under this model, each number is represented as a signed integer coded on $(1 + n + m)$ bits, where one bit is used for the sign, $n$ bits are used for the integral part of the number, and $m$ bits are used for its fractional part. Using this model, we can write a given number $z$ as

$$z = (-1)^{z_n} \sum_{b=-m}^{n-1} 2^b z_b, \tag{3.8}$$

where $z_b \in \{0, 1\}$ are the bits stored in memory to represent $z$. In our modeling of the Kalman filter, all variables (including matrix components) involved in Equations (3.3)–(3.7) are stored using this quantization model, all with the same values of $n$ and $m$. The quantization of the variables to this fixed-point model is done using a uniform quantizer. Note that the distribution of the quantized data is not necessarily uniform (the random variables $\hat{\boldsymbol{x}}_{k|k}$ and $\boldsymbol{y}_k$ could follow Gaussian distributions for example). However, in [128] it is shown that a uniform quantizer can be applied independently of the probability distribution of the source with only a small difference to an optimal quantizer.

In the considered quantizer, the value of $n$ is chosen to be able to represent the largest possible value in the system. The value of $m$ sets the resolution of the quantization so that the smallest difference between two quantized numbers is $2^{-m}$ [14]. The value of $m$ will be a parameter that is optimized for minimizing the energy in later sections.

In the case of fixed values, such as components of the matrices of the filter, the fixed-point quantized value can be written as $\underline{f} = f + \delta_f$ where $\delta_f$ is the quantization error. Using the previously described uniform quantizer, $\delta_f < 2^{-m}$. In the case of quantized random variables, such as the components of $\hat{\boldsymbol{x}}_{\boldsymbol{k}|\boldsymbol{k}}$ or $\boldsymbol{y}_{\boldsymbol{k}}$, we let $\epsilon_x$ be the quantization error and express $\underline{x} = x + \epsilon_x$. In [129], conditions are given for the quantization error $\epsilon_x$ to be independent from the quantized variable depending on the distribution of the quantized data.

For the special case of a Gaussian distribution, the quantization step needs to be significantly smaller than the variance of the quantized data. In this case, it can be shown that the quantization error is a white noise following a uniform distribution of variance $\frac{2^{-2m}}{12}$ [129]. This independence assumption will be used in the theoretical derivations of Section 3.3. Note that most existing works on quantized Kalman filters only consider that random quantities, such as $\hat{\boldsymbol{x}}_{\boldsymbol{k}|\boldsymbol{k}}$ and $\boldsymbol{y}_{\boldsymbol{k}}$, are quantized, whereas here, the components of the matrices, e.g., $\boldsymbol{K}_k$ of the filter, are also quantized. This will require a new theoretical analysis to treat this case.

### 3.2.3 Implementation of the Filter by Using an Unreliable Memory

In order to reduce its energy consumption, the Kalman filter can be implemented on unreliable hardware [15,91,92,98]. Here, we assume, as in [92,98], that only the memory is faulty. In this case, each memory cell of a memory bank has a bit flipping probability $p$. We then use the model of [28] to express $p$ with respect to the memory bank energy consumption $e$ as

$$p = \exp(-ea)\,, \tag{3.9}$$

where $a$ is a parameter that depends on the device technology. We assume that bit errors occur independently. This is justified first by the fact that, in many cases of interest, such as the common case of SRAM memories in a CMOS digital circuit, memory failures can be assumed to occur independently for each bit cell [130]. Therefore, we have a spatial independence between each memory cell for one iteration. However, typically faults are caused by fabrication variations, and therefore this cannot guarantee a temporal independence for successive reads of the same memory cells. To resolve this issue, we can assume that a diversity scheme is implemented at the system level to avoid re-using the same memory location to store the same variable, which can be implemented at very low cost simply by modifying the memory addressing scheme.

Each memory bank has a uniform energy consumption (e.g., single supply voltage) and is used in our case to store the bits at a certain position of all components of matrices that are stored in the unreliable memory. Since the other terms of the filter can be precomputed offline and stored on a reliable memory separately in the system, we assume that only the estimates $\hat{\boldsymbol{x}}_{k+1|k}$ and $\hat{\boldsymbol{x}}_{k+1|k+1}$ are stored in an unreliable memory bank. Therefore, in the Kalman filter, instead of having an estimate component $\hat{x}$, such as the one computed in (3.3), we have a possibly incorrect estimate component $\tilde{x}$. We can define an energy per memory bank vector using the binary representation given in (3.8):

$$\boldsymbol{e} = \left[e_{-m}, e_{-m+1}, \ldots, e_{n-1}\right] . \tag{3.10}$$

We can then express as $\tilde{x}_b = \hat{x}_b \oplus \gamma_b$ a bit at position $b$ stored in the unreliable memory. Here, $p_b = \Pr(\gamma_b = 1) = \exp(-e_b a)$, and $\oplus$ denotes the modulo-2 addition. As the filter would be particularly sensitive to faults on the sign bit, we consider a sign-preserving model, as in [92, 131, 132]. This sign-preserving model can be implemented by storing the sign bits in a separate reliable memory.

Using this noise model defined at the bit-level $\tilde{x}_b$, we can define a noise model at the symbol level $\tilde{x}$ as

$$\tilde{x} = \hat{x} + \gamma \,, \tag{3.11}$$

where $\gamma$ is the noise introduced by the unreliable memory. For the subsequent theoretical analysis, we assume that the mean $\mathbb{E}[\gamma]$ of this memory noise is negligible compared to its variance $\mathrm{Var}[\gamma] = \sigma_\gamma^2$. We verified this condition with Monte Carlo simulations. The covariance matrix $\boldsymbol{\Gamma}$ of a memory noise vector $\boldsymbol{\gamma}$ of length $c$ is defined as $\boldsymbol{\Gamma} = \mathrm{Cov}[\boldsymbol{\gamma}] = \boldsymbol{I}_c \sigma_\gamma^2$, and has size $c \times c$. The matrix $\boldsymbol{\Gamma}$ is diagonal since the memory noise variables are considered independent.

### 3.3 Error Analysis

As described in Section 3.2, we consider two types of errors affecting the filter: the quantization error and the unreliable memory noise. In this section, we first describe a generic model of error propagation in the Kalman filter, before studying both types of errors in more detail. Finally, we compute the new covariance matrix $\boldsymbol{P}^*_{\boldsymbol{k|k}} = \mathrm{Cov}[\tilde{\boldsymbol{x}}_{k|k} - \boldsymbol{x}_k]$ of the total estimation error by taking both sources of noise (quantization and unreliable memories) into account, compared to a standard Kalman filter, which does not include either.

### 3.3.1 Error Propagation Model

Our objective is to compute the total error $\boldsymbol{\Delta\hat{x}_{k+1|k+1}}$ on the computation of $\hat{\boldsymbol{x}}_{k+1}$ at step $k+1$ by considering the two types of errors: quantization and unreliable memory. To handle recursion as in [120], we choose to split the error model in two parts: the errors occurring at step $k$ and the errors from the previous steps, which are propagated up to step $k$.

To compute $\boldsymbol{\Delta\hat{x}_{k+1|k+1}}$, we first need to express the total error $\boldsymbol{\Delta P}_{k+1|k}$ on the a posteriori covariance matrix $\boldsymbol{P}_{k+1|k}$ after step $k+1$. As in [120], we express this total error as

$$\boldsymbol{\Delta P_{k+1|k}} = f_P(\boldsymbol{\Delta P_{k|k-1}}) + \boldsymbol{\delta P_{k+1|k}}\,, \tag{3.12}$$

where the function $f_P$ models the errors propagated from step $k$, and $\boldsymbol{\delta P_{k+1|k}}$ represents the errors occurring at step $k+1$. In this case, according to [120]:

$$f_P(\boldsymbol{\Delta P_{k|k-1}}) = \boldsymbol{G}_k\boldsymbol{\Delta P_{k|k-1}}\boldsymbol{G_k^\top} + o(\boldsymbol{\Delta^2})\,, \tag{3.13}$$

where $\boldsymbol{\Delta} = \left\|\boldsymbol{H\Delta P_{k|k-1}H^\top}\right\|_2 \ll \sigma_{\min}(\boldsymbol{HP}_{k|k-1}\boldsymbol{H}^\top + \boldsymbol{R})$ and $\boldsymbol{HP}_{k|k-1}\boldsymbol{H}^\top + \boldsymbol{R}$ is a square nonsingular matrix with $\sigma_{\min}$ representing the smallest singular value. Therefore, we have the approximation

$$f_P(\boldsymbol{\Delta P_{k|k-1}}) \approx \boldsymbol{G}_k\boldsymbol{\Delta P_{k|k-1}}\boldsymbol{G_k^\top}\,, \tag{3.14}$$

where $\boldsymbol{G}_k = \boldsymbol{F}(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})$. We then express the total error $\boldsymbol{\Delta\hat{x}_{k+1|k+1}}$ on $\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}$ by considering the same separation between propagation errors and errors from the current iteration. This gives

$$\boldsymbol{\Delta\hat{x}_{k+1|k+1}} = f_x(\boldsymbol{\Delta\hat{x}_{k|k}}, \boldsymbol{\Delta P_{k|k-1}}) + \boldsymbol{\delta\hat{x}_{k+1|k+1}}\,, \tag{3.15}$$

where the error propagation function $f_x$ is provided in [120], using the same assumption as for (3.13), as

$$\begin{aligned} &f_x(\boldsymbol{\Delta\hat{x}_{k|k}}, \boldsymbol{\Delta P_{k|k-1}}) \approx \\ &\quad (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})(\boldsymbol{F\Delta\hat{x}_{k|k}} + \boldsymbol{\Delta P_{k|k-1}}\boldsymbol{H}^\top(\boldsymbol{HP}_{k,k-1}\boldsymbol{H}^\top + \boldsymbol{R})^{-1}(\boldsymbol{y}_{k+1} - \boldsymbol{HF}\hat{\boldsymbol{x}}_{k,k}))\,. \end{aligned} \tag{3.16}$$

In this expression, we observe the error propagation from the previous computations of $\hat{\boldsymbol{x}}_{k|k}$ and $\boldsymbol{P}_{k+1|k}$. In particular, $\hat{\boldsymbol{x}}_{k+1|k+1}$ depends on $\boldsymbol{K}_{k+1}$, which is precomputed from $\boldsymbol{P}_{k+1|k}$ at each iteration.

Using the recursive Equations (3.12) and (3.15), we now estimate the covariance matrix of the total estimation error $\boldsymbol{P^*_{k|k}}$. Note that [120] considered only quantization errors, while here

we consider two sources of errors: quantization and unreliable memories. To evaluate $\boldsymbol{P}^*_{\boldsymbol{k|k}}$, we must first compute the covariance of each term of $\boldsymbol{\Delta\hat{x}_{k+1|k+1}}$. By assuming that the two sources of noise (quantization and unreliable memory noise) are statistically independent, we decompose $\boldsymbol{\delta\hat{x}_{k+1|k+1}}$ as

$$\boldsymbol{\delta\hat{x}_{k+1|k+1}} = \boldsymbol{\delta\hat{x}^{quant}_{k+1|k+1}} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}}\,, \tag{3.17}$$

and study the two terms $\boldsymbol{\delta\hat{x}^{quant}_{k+1|k+1}}$ and $\boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}}$ separately.

### 3.3.2 Quantization Error

We now aim for an analytical expression for $\boldsymbol{\delta\hat{x}^{quant}_{k+1|k+1}}$, defined as the difference between the full precision estimate $\hat{\boldsymbol{x}}_{k+1|k+1}$ and its quantized version $\underline{\hat{\boldsymbol{x}}}_{k+1|k+1}$:

$$\boldsymbol{\delta\hat{x}^{quant}_{k+1|k+1}} = \hat{\boldsymbol{x}}_{k+1|k+1} - \underline{\hat{\boldsymbol{x}}}_{k+1|k+1}\,. \tag{3.18}$$

Before expressing $\boldsymbol{\delta\hat{x}^{quant}_{k+1|k+1}}$, we first review generic quantization errors expressions [120]. For the scalar fixed-point multiplication of a coefficient $\underline{s}$ with a random variable $\underline{t}$ both quantized according to the model presented in Section 3.2.2, we can show that

$$\begin{aligned}\underline{st} &= (s+\delta_s)(t+\epsilon_t) + \epsilon_{st} \\ &= st + s\epsilon_t + t\delta_s + \delta_s\epsilon_t + \epsilon_{st}\,,\end{aligned} \tag{3.19}$$

where $\delta_s = \underline{s} - s$ and $\epsilon_t$ and $\epsilon_{st}$ follow uniform distributions of variance $\frac{2^{-2m}}{12}$. The scalar expression (3.19) can then be generalized to the case of a product between a matrix of fixed-point coefficients $\underline{\boldsymbol{A}}$ of size $p \times q$ and a matrix of fixed-point random variables $\underline{\boldsymbol{B}}$ of size $q \times r$ as

$$\underline{\boldsymbol{AB}} = \boldsymbol{AB} + \boldsymbol{A\epsilon_B} + \boldsymbol{B\delta_A} + \boldsymbol{\delta_A\epsilon_B} + \boldsymbol{\epsilon_{AB}}\,, \tag{3.20}$$

where $\boldsymbol{\epsilon_{AB}}$ is of size $p \times r$ with $\epsilon_{ABi,j} = \sum_{k=1}^{q} \epsilon_{ABi,j,k}$. According to Section 3.2.2, each $\epsilon_{ABi,j,k}$ follows a uniform distribution of variance $\frac{2^{-2m}}{12}$. In (3.20), the product $\boldsymbol{\delta_A\epsilon_B}$ can be considered as negligible compared to the other error terms. Indeed, all scalar quantization errors $\epsilon$ and $\delta$ are upper-bounded by $2^{-m-1}$, and since $m \geq 1$, their product is bounded by $2^{-2m-2}$. Thus, given that, for a value of $m$ large enough, the value of $2^{-m-1}$ is much less than 1 and $2^{-2m-2} = (2^{-m-1})^2$, we have that $2^{-2m-2}$ is negligible compared to $2^{-m-1}$. Therefore, in the following derivation, we neglect the products of quantization errors.

We now study quantization errors introduced during the computation of $\underline{\hat{\boldsymbol{x}}}_{k+1|k+1}$. While

existing works, e.g., [121–123], assume that only the random quantities $\underline{\hat{\boldsymbol{x}}}_{k|k}$ and $\underline{\boldsymbol{y}}_{k+1}$ are quantized, we here also consider that the matrices $\underline{\boldsymbol{D}}_{k+1}$ and $\underline{\boldsymbol{K}}_{k+1}$ are quantized as well. This corresponds to a more practical implementation setup and requires a more complex theoretical analysis. We first note that Equation (3.6) can be rewritten as

$$\underline{\hat{\boldsymbol{x}}}_{k+1|k+1} = \underline{\boldsymbol{D}}_{k+1}\underline{\hat{\boldsymbol{x}}}_{k|k} + \underline{\boldsymbol{K}}_{k+1}\underline{\boldsymbol{y}}_{k+1}\,, \tag{3.21}$$

where both $\boldsymbol{D}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{F}$ and the Kalman gains $\boldsymbol{K}_k$ can be computed offline. We thus consider that the matrices $\boldsymbol{K}_k$ and $\boldsymbol{D}_k$ are computed in full precision and then quantized with a fixed point model. Under these conditions, according to (3.20) and if we consider that the product of quantization errors is negligible, the quantized vector $\underline{\hat{\boldsymbol{x}}}_{k+1|k+1}$ can be approximated as

$$\begin{aligned}\underline{\hat{\boldsymbol{x}}}_{k+1|k+1} = {} & \boldsymbol{D}_{k+1}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} \\ & + \boldsymbol{\epsilon}_{\boldsymbol{D}_{k+1}\boldsymbol{x}_{k|k}} + \boldsymbol{K}_{k+1}\boldsymbol{y}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}}\boldsymbol{y}_{k+1} \\ & + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} + \boldsymbol{\epsilon}_{\underline{\boldsymbol{K}}_{k+1}\underline{\boldsymbol{y}}_{k+1}} + o(2^{-m-1})\,.\end{aligned} \tag{3.22}$$

We see that the expression of $\underline{\hat{\boldsymbol{x}}}_{k+1|k+1}$ depends on the full precision vectors $\hat{\boldsymbol{x}}_{k|k}$ and $\boldsymbol{y}_{k+1}$ and on the quantization errors and noise. Finally, the quantization error $\boldsymbol{\delta\hat{x}}^{\mathbf{quant}}_{\boldsymbol{k+1|k+1}}$ defined in (3.18) can be computed by using (3.22):

$$\begin{aligned}\boldsymbol{\delta\hat{x}}^{\mathbf{quant}}_{\boldsymbol{k+1|k+1}} \approx {} & \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} + \boldsymbol{\epsilon}_{\boldsymbol{D}_{k+1}\boldsymbol{x}_{k|k}} \\ & + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}}\boldsymbol{y}_{k+1} + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} + \boldsymbol{\epsilon}_{\boldsymbol{K}_{k+1}\boldsymbol{y}_{k+1}},\end{aligned} \tag{3.23}$$

where the covariance matrix $\boldsymbol{\Sigma}_\times$ of $\boldsymbol{\epsilon}_\times = \boldsymbol{\epsilon}_{\boldsymbol{D}_{k+1}\boldsymbol{x}_{k|k}} + \boldsymbol{\epsilon}_{\boldsymbol{K}_{k+1}\boldsymbol{y}_{k+1}}$ is given by

$$\boldsymbol{\Sigma}_\times = \mathrm{Cov}[\boldsymbol{\epsilon}_{\boldsymbol{D}_{k+1}\boldsymbol{x}_{k|k}}] + \mathrm{Cov}[\boldsymbol{\epsilon}_{\underline{\boldsymbol{K}}_{k+1}\underline{\boldsymbol{y}}_{k+1}}] = \mathbf{I}_c(c+d)\frac{2^{-2m}}{12}, \tag{3.24}$$

and $\mathrm{Cov}[\boldsymbol{\epsilon}_{x_{k|k}}] = \mathbf{I}_c\frac{2^{-2m}}{12}$, $\mathrm{Cov}[\boldsymbol{\epsilon}_{y_{k+1}}] = \mathbf{I}_d\frac{2^{-2m}}{12}$. Equation (3.23) gives us the quantization error on the computation of $\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}}$ based on the unquantized values of $\hat{\boldsymbol{x}}_{\boldsymbol{k|k}}$, the filter parameters and the quantization resolution $m$.

### 3.3.3 Unreliable Memory Error

We now consider the second source of noise from the unreliable memories and derive an expression for the covariance matrix $\boldsymbol{\Gamma} = \mathrm{Cov}[\boldsymbol{\delta\hat{x}}^{\mathbf{mem}}_{\boldsymbol{k+1|k+1}}]$ of the unreliable memory noise $\boldsymbol{\delta\hat{x}}^{\mathbf{mem}}_{\boldsymbol{k+1|k+1}}$ introduced in (3.15).

Assuming $\mathbb{E}[\gamma] \ll \mathrm{Var}[\gamma]$, as discussed in Section 3.2.3, the variance $\mathrm{Var}[\gamma] = \sigma_\gamma^2$ of the memory noise $\gamma$ can be approximated by the MSE as $\sigma_\gamma^2 \approx \mathbb{E}[(\tilde{x} - \hat{x})^2]$. The value of $\mathbb{E}[(\tilde{x} - \hat{x})^2]$ depends on the error probabilities $p_b$ as well as on the probability distributions of the variables $x$, which are stored in memory. However, from ([15], Claim 17), if the probability of error on the most significant bit $p_{n-1} \ll \frac{1}{2}$ or $\Pr(\hat{x}_b = \hat{x}_{b'}) \simeq \Pr(\hat{x}_b \neq \hat{x}_{b'})$ for any $b \neq b'$, then the MSE $\mathbb{E}[(\tilde{x} - \hat{x})^2]$ can be approximated as

$$\sigma_\gamma^2 = \mathbb{E}[(\tilde{x} - \hat{x})^2] \approx \sum_{b=-m}^{n-1} 4^b p_b = \sum_{b=-m}^{n-1} 4^b e^{-e_b a}\,, \tag{3.25}$$

where the last equality is obtained from the noise-versus-energy model (3.9). Therefore, the probability distributions of the variables $x$ have no significant impact on the value of the MSE.

Equation (3.25) gives us a relation between the noise variance $\sigma_\gamma^2$ and the vector $\boldsymbol{e}$ of energy levels defined in (3.10). Moreover, by using (3.25), we show that the covariance $\boldsymbol{\Gamma}$ of the memory noise vector $\boldsymbol{\delta\hat{x}^{\mathrm{mem}}_{k+1|k+1}}$ is given by

$$\boldsymbol{\Gamma} = \boldsymbol{I}_c \sigma_\gamma^2\,. \tag{3.26}$$

### 3.3.4 Total Error

After separately studying the two error terms $\boldsymbol{\delta x^{\mathrm{quant}}_{k+1|k+1}}$ and $\boldsymbol{\delta\hat{x}^{\mathrm{mem}}_{k+1|k+1}}$, we now combine them to get an expression of the total estimation error $\boldsymbol{e^*_{k+1|k+1}} = \tilde{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1}$. We then provide the covariance matrix $\boldsymbol{P^*_{k+1|k+1}}$ of this total error.

By using $\tilde{\boldsymbol{x}}_{k|k}$ to denote the faulty estimate of $\boldsymbol{x}_k$, we can express

$$\tilde{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k} + \boldsymbol{\Delta\tilde{x}_{k|k}}\,. \tag{3.27}$$

Considering that only the $\hat{\boldsymbol{x}}_{k|k}$ are stored in the unreliable memories, the error propagation model (3.15) can be rewritten as

$$\boldsymbol{\Delta\tilde{x}_{k+1|k+1}} = \boldsymbol{D}_{k+1}\boldsymbol{\Delta\tilde{x}_{k|k}} + \boldsymbol{\delta\hat{x}^{quant}_{k+1|k+1}} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}} \tag{3.28}$$

$$\begin{aligned} &= \boldsymbol{D}_{k+1}\boldsymbol{\Delta\tilde{x}_{k|k}} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}}\tilde{\boldsymbol{x}}_{k|k} + \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} \\ &\quad + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}}\boldsymbol{y}_{k+1} + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} \\ &\quad + \boldsymbol{\epsilon}_{\times} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}} \end{aligned} \tag{3.29}$$

$$\begin{aligned} &= (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\boldsymbol{\Delta\tilde{x}_{k|k}} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}}\hat{\boldsymbol{x}}_{k|k} \\ &\quad + \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}}\boldsymbol{y}_{k+1} + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} \\ &\quad + \boldsymbol{\epsilon}_{\times} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}}, \end{aligned} \tag{3.30}$$

where (3.29) is obtained by replacing $\boldsymbol{\delta\hat{x}^{quant}_{k+1|k+1}}$ by its expression (3.23), and (3.30) comes from (3.27), which allows u to write $\tilde{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k} + \boldsymbol{\Delta\tilde{x}_{k|k}}$. Equation (3.30) provides a recursive form of the error at step $k$, since $\boldsymbol{\Delta\tilde{x}_{k+1|k+1}}$ depends on $\boldsymbol{\Delta\tilde{x}_{k|k}}$ and $\hat{\boldsymbol{x}}_{k|k}$. All the other terms in (3.30) come from the current iteration $k+1$.

In certain conditions, such as if $\boldsymbol{H}$ and $\boldsymbol{F}$ are only composed of integer components, the total estimation error $\tilde{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1} = \hat{\boldsymbol{x}}_{k+1|k+1} + \boldsymbol{\Delta\tilde{x}_{k+1|k+1}} - \boldsymbol{x}_{k+1}$ can be further developed as :

$$\tilde{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1} = \hat{\boldsymbol{x}}_{k+1|k+1} + \boldsymbol{\Delta\tilde{x}_{k+1|k+1}} - \boldsymbol{x}_{k+1} \tag{3.31}$$

$$\begin{aligned} &= (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\boldsymbol{\Delta\tilde{x}_{k|k}} + (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} \\ &+ (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{y}_{k+1} + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} + \boldsymbol{\epsilon}_{\times} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}} - \boldsymbol{x}_{k+1} \end{aligned} \tag{3.32}$$

$$\begin{aligned} &= (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\boldsymbol{\Delta\tilde{x}_{k|k}} + (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} \\ &+ (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})(H\boldsymbol{x}_{k+1} + \boldsymbol{v}_{k+1}) + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} + \boldsymbol{\epsilon}_{\times} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}} - \boldsymbol{x}_{k+1} \end{aligned} \tag{3.33}$$

$$\begin{aligned} &= (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\boldsymbol{\Delta\tilde{x}_{k|k}} + (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} \\ &+ (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{v}_{k+1} + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} + \boldsymbol{\epsilon}_{\times} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}} \\ &+ ((\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})H - \mathbf{I})\boldsymbol{x}_{k+1} \end{aligned} \tag{3.34}$$

$$\begin{aligned} &= (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\tilde{\boldsymbol{x}}_{k|k} - (\mathbf{I} - (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{H})\boldsymbol{F}\boldsymbol{x}_k \\ &+ (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{v}_{k+1} + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} + \boldsymbol{\epsilon}_{\times} + \boldsymbol{\delta\hat{x}^{mem}_{k+1|k+1}} \\ &+ \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} + ((\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})H - \mathbf{I})\boldsymbol{u}_k\,. \end{aligned} \tag{3.35}$$

If $\boldsymbol{H}$ and $\boldsymbol{F}$ are only composed of integer components, due to how the quantization is done:

$$(\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}}) = (\mathbf{I} - (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{H})\boldsymbol{F}\,. \tag{3.36}$$

And therefore:

$$\tilde{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1} = (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})(\tilde{\boldsymbol{x}}_{k|k} - \boldsymbol{x}_k) \tag{3.37}$$

$$+ (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{v}_{k+1} + ((\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{H} - \mathbf{I})\boldsymbol{u}_k \tag{3.38}$$

$$+ \boldsymbol{D}_{k+1}\boldsymbol{\epsilon}_{x_{k|k}} + \boldsymbol{K}_{k+1}\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}} + \boldsymbol{\epsilon}_\times + \boldsymbol{\delta}\hat{\boldsymbol{x}}^{\mathbf{mem}}_{\boldsymbol{k+1|k+1}}. \tag{3.39}$$

This equation gives us a recursive form of the total estimation error $\tilde{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1}$ at step $k+1$, depending on the estimation error $\tilde{\boldsymbol{x}}_{k|k} - \boldsymbol{x}_k$ at step $k$ and on the quantization resolution $2^{-m}$ and the memory noise $\boldsymbol{\delta}\hat{\boldsymbol{x}}^{\mathbf{mem}}_{\boldsymbol{k+1|k+1}}$.

Finally, we can compute the covariance matrix $\boldsymbol{P}^*_{k+1|k+1}$ of this error as

$$\begin{aligned}
\boldsymbol{P}^*_{k+1|k+1} &= \mathrm{Cov}[\tilde{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1}] \\
&= (\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})\boldsymbol{P}^*_{k|k}(\boldsymbol{D}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{D_{k+1}}})^\top \\
&+ (\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{R}(\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})^\top \\
&+ ((\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{H} - \mathbf{I})\boldsymbol{Q}((\boldsymbol{K}_{k+1} + \boldsymbol{\delta}_{\boldsymbol{K_{k+1}}})\boldsymbol{H} - \mathbf{I})^\top \\
&+ \boldsymbol{D}_{k+1}\,\mathrm{Cov}[\boldsymbol{\epsilon}_{x_{k|k}}]\boldsymbol{D}^\top_{k+1} + \boldsymbol{K}_{k+1}\,\mathrm{Cov}[\boldsymbol{\epsilon}_{\boldsymbol{y_{k+1}}}]\boldsymbol{K}^\top_{k+1} + \boldsymbol{\Sigma}_\times + \boldsymbol{\Gamma},
\end{aligned} \tag{3.40}$$

where all the terms involved, including the covariance matrices, have been explicited in the previous sections. Equation (3.40) shows that the covariance matrix $\boldsymbol{P}^*_{k+1|k+1}$ can be computed recursively.

Equation (3.40) provides a measure of the performance of the filter, depending on the quantization resolution and on the energy supplied to the memory. Equipped with this derivation, we can now use the covariance matrix $\boldsymbol{P}^*_{k+1|k+1}$ as a performance criterion against which to optimize the energy consumed by the unreliable memory.

### 3.4 Energy Optimization

In this section, we optimize the energy consumption of the memory while satisfying a performance constraint defined on the total estimation error of the filter. As parameters to optimize, we consider the number of bits $m$ for the quantization, and the energy vector $\boldsymbol{e}$ of the memory banks. We define two optimization problems, which both seek to minimize the energy consumed by the memory. In the first problem, we find the optimal number of bits $m$ and the corresponding $n+m$ levels of energy to allocate to the memory banks. Although solving this problem provides the minimum energy that needs to be supplied to the memory, it is not very practical since each of the $n+m$ bits should be stored in a different memory bank with a specific voltage supply.

Therefore, in the second problem, we consider that the number of bits $m$ is fixed but that the number of possible energy levels is limited to $L$ possibilities. Both the $L$ energy values and the allocation of each bit to one of the $L$ possible values should be optimized. Solving this problem allows to consider only $L < n + m$ different memory banks.

### 3.4.1 Optimization across All the Bits

We first find the optimal level of energy $e_b$ of each memory bank and the optimal number of fractional bits $m$ to minimize the total memory energy consumption. As performance criterion, we consider the covariance matrix $\boldsymbol{P}^*_{N|N}$ of the total estimation error at step $N$, where $N$ is chosen to be large enough so that the filter can converge. We further introduce a matrix $\boldsymbol{\mathcal{V}}$ of the same size as $\boldsymbol{P}^*_{N|N}$ to define the performance constraint for the variances and covariances of estimation error on each component. The optimization problem is then defined as follows:

$$\begin{aligned} \min_{\mathbf{e},m} \quad & e_{\text{tot}} = \sum_{b=-m}^{n-1} e_b = \mathbb{1}^\top \boldsymbol{e}\,, \\ \text{s.t.} \quad & \boldsymbol{P}^*_{N|N} \prec \boldsymbol{\mathcal{V}} \text{ and } e_b \geq e_{\text{thres}}\ \forall b \in [\![-m, n-1]\!]\,, \end{aligned} \tag{3.41}$$

where $\prec$ is a component-wise inequality between the two matrices and where the minimum is taken over all energy vectors $\boldsymbol{e}$ as defined in (3.10) and for all the possible values of the number of bits $m$. We consider that $m \in [\![0, M]\!]$, where $M$ is the maximum number of bits, which could be stored in a memory. The value $e_{\text{thres}}$ is the minimum level of energy for each memory bank to avoid undesired effects, such as circuit delays and energy leakage [28].

Problem (3.41) involves one discrete parameter $m$ and $m+n$ continuous parameters $\boldsymbol{e}$, which makes it difficult to solve at once. As a first step, we assume that the value of $m$ is fixed and solve the following simplified problem:

$$\begin{aligned} \min_{\mathbf{e}} \quad & e_{\text{tot}} = \sum_{b=-m}^{n-1} e_b = \mathbb{1}^\top \boldsymbol{e}\,, \\ \text{s.t. } & \boldsymbol{P}^*_{N|N} \prec \boldsymbol{\mathcal{V}} \quad \text{and} \quad e_b \geq e_{\text{thres}}\ \forall b \in [\![-m, n-1]\!]\,, \end{aligned} \tag{3.42}$$

by using the Karush–Kuhn–Tucker (KKT) conditions.

From optimization Problem (3.42), we can define the Lagrangian:

$$L(e, \nu, \lambda) = \sum_{b=0}^{B-1} e_b + \nu\Big(\sum_{b=0}^{B-1} 4^b e^{-e_b a} - \mathcal{V}\Big) - \sum_{b=0}^{B-1} \lambda_b (e_b - e_{\text{thres}})\,. \tag{B1}$$

From the KKT conditions, for the optimal solution $\boldsymbol{e}^*$:

$$\nu(\sum_{b=0}^{B-1} 4^b e^{-e_b^* a} - \mathcal{V}) = 0 \ , \nu \geq 0 \tag{B2}$$

$$\lambda_b(e_b^* - e_{\text{thres}}) = 0 \ , \lambda_b \geq 0 \ \forall b \in [\![0, B-1]\!] \tag{B3}$$

$$\frac{\partial L}{\partial e_b^*} = 1 - \nu 4^b a e^{-e_b a} - \lambda_b = 0 \tag{B4}$$

From (B3) and (B4):

$$\lambda_b = 1 - \nu 4^b a e^{-e_b a} \geq 0 \, . \tag{B5}$$

If $\nu = 0$ then $\lambda_b = 1$ and $e_b = e_{\text{thres}}$. Therefore, we claim that $\nu \neq 0$, and thus $\sum_{b=0}^{B-1} 4^b e^{-e_b^* a} = \mathcal{V}$.

If $\nu \leq \frac{1}{4^b a}$, then it is not possible to have $e_b > e_{\text{thres}}$ since it would mean that $\lambda_b = 0$, and thus $\nu = \frac{1}{4^b a} e^{e_b a} \geq \frac{1}{4^b a}$, which is in contradiction with the hypothesis. Therefore, if $\nu \leq \frac{1}{4^b a}$, then $e_b = e_{\text{thres}}$.

If $\nu > \frac{1}{4^b a}$, then by the same logic as before $e_b > e_{\text{thres}}$. In this case, $\lambda_b = 0$, and thus $e_b = \frac{1}{a} \log(\nu 4^b a)$.

From these conditions, we show that the optimal energy level $e_b^*$ for bit $b$ has expression

$$e_b^* = \begin{cases} e_{\text{thres}}, & \text{if } \lambda < \frac{1}{4^b a} \, , \\ \frac{1}{a} \log(4^b a \lambda), & \text{otherwise} \, , \end{cases} \tag{3.43}$$

where $\lambda$ is a dual variable. It allows balancing a trade-off between preserving the performance of the system and reducing the energy consumption. A water-filling algorithm [15] can be used to compute the optimal vector $\boldsymbol{e}^*$ for a fixed desired performance $\boldsymbol{\mathcal{V}}$ of the filter. We can observe that, according to this optimal solution, the energy of the least significant bits will be set to the threshold energy level $e_{\text{thres}}$. The energy levels then increase logarithmically for each bit as their significance increases.

Since $m$ is discrete, the optimal solution (3.43) is computed using the water-filling algorithm for each possible value of $m$. We then retain the solution $(m^\star, \boldsymbol{e}^\star)$, which gives the lowest total energy $e_{\text{tot}}^\star = \sum_{b=-m^\star}^{n} e_b^\star$. In this method, the influence of the quantization error is taken into account through the performance criterion $\boldsymbol{P}_{N|N}^*$. For a small number of bits $m$, quantization errors may make it impossible to satisfy the desired performance constraint, and therefore the water-filling algorithm will not be able to find an optimal solution. In this case, if we detect that the algorithm converges toward a performance value that is still higher

than the constraint, the algorithm is stopped, and we proceed to the next value of $m$ in the considered range.

The full optimization process is summarized in Algorithm 1. In this algorithm, the parameter $\beta$ controls the rate at which the energy for each memory bank is increased at each iteration. The value of $\beta$ is chosen either using the precision with which energy can be set in a given device technology, or based on the desired rate of convergence for the water-filling algorithm.

The condition $\left\|\boldsymbol{P}_{\text{prev}} - \boldsymbol{P}^*_{N|N}\right\| > \xi$ is used to detect whether the water-filling algorithm has a feasible solution, and thus the value of $\xi$ is set to be low. The computation of $\boldsymbol{P}^*_{N|N}(\boldsymbol{e}, m)$ accounts for most of the computing time of this algorithm. The total run time thus depends on the number of iterations required by the water-filling algorithm (while loop in Algorithm 1). For fixed values of $\beta$ and $\boldsymbol{\mathcal{V}}$, we expect the number of iterations to increase with $m$.

**Algorithm 1:** Computing the optimal values for $\boldsymbol{e}$ and $m$.

**Input:** $\boldsymbol{\mathcal{V}}$, $a$, $\beta$, $\xi$, $e_{\text{thres}}$ ;
$\boldsymbol{e}_{\min} \leftarrow +\infty$;
**for** *each value of* $m$ **do**
  $\boldsymbol{e} \leftarrow e_{\text{thres}}$ ;
  $\boldsymbol{P}_{\text{prev}} \leftarrow 0$;
  **while** $\boldsymbol{P}^*_{N|N}(\boldsymbol{e}, m) \succ \boldsymbol{\mathcal{V}}$ *and* $\left\|\boldsymbol{P}_{prev} - \boldsymbol{P}^*_{N|N}\right\| > \xi$ **do**
    $\boldsymbol{P}_{\text{prev}} \leftarrow \boldsymbol{P}^*_{N|N}(\boldsymbol{e}, m)$;
    $b \leftarrow \arg\min_b \{\log(\frac{1}{4^b a}) + ea\}$;
    $e_b \leftarrow e_b + \beta$ ;
  **end**
  **if** $\boldsymbol{P}^*_{N|N}(\boldsymbol{e}, m) \prec \boldsymbol{\mathcal{V}}$ *and* $\sum_{b=-m_{opti}}^{n} e_{min_b} > \sum_{b=-m}^{n} e_b$ **then**
    $\boldsymbol{e}_{\min} \leftarrow \boldsymbol{e}$ ;
    $m_{\text{opti}} \leftarrow m$ ;
  **end**
**end**
**Result:** Optimal number of bits $m_{\text{opti}}$ and optimal energy allocation vector $\boldsymbol{e}_{\min}$

### 3.4.2 Optimization with a Limited Number of Energy Levels

In practice, the solution of Problem 1 makes the implementation costly as each bit position should be stored in a separate memory bank. Therefore, we define a second optimization problem with only $L < m + n$ possible levels of energy. For implementation purposes, we only consider small values for $L$ ($L < 10$). The vector $\boldsymbol{f} = [f_0, \ldots, f_{L-1}]$ contains the $L$ levels of energy. We use $n_\ell$ to denote the number of bits allocated to energy level $f_l$, so that

$\sum_{\ell=0}^{L-1} n_\ell = n + m$. This means that each memory bank of the energy group $\ell$ has an energy level $e_b = \frac{f_\ell}{n_\ell}$. We write $\boldsymbol{n} = [n_0, \ldots, n_{L-1}]$ for the vector containing the $L$ values $n_\ell$.

In the following, for simplicity, we consider that the number of bits $n$ and $m$ are fixed, and we seek to optimize the total energy consumption of the unreliable memory for a fixed number of energy levels $L$. The objective is to reduce the total energy consumed by the unreliable memory by allocating different levels of energy to the $L$ groups of bits. Two parameters are considered in this optimization: the values of each energy level $e_\ell$ and the number of bits allocated to each of these energy levels $n_\ell$. The optimization problem can be written as

$$\begin{aligned} \min_{\boldsymbol{f},\boldsymbol{n}} \quad & e_{\text{tot}} = \sum_{l=0}^{L-1} f_\ell = \mathbb{1}^\top \boldsymbol{f}\,, \\ \text{s.t.} \quad & \boldsymbol{P}^*_{N|N} \prec \boldsymbol{\mathcal{V}} \quad \text{and} \qquad f_\ell \geq e_{\text{thres}} n_\ell,\ n_\ell \in [\![1, n+m-L]\!]\, \forall \ell \in [\![0, L-1]\!]\,. \end{aligned} \tag{3.44}$$

First, we solve the optimization problem in the case where we know which bit is allocated to which energy level. This means that the values of $n_\ell$ are known and that we only want to compute the optimal values of the energy levels $f_\ell$. In this case, the optimization problem can be written as

$$\begin{aligned} \min_{\boldsymbol{f}} \quad & e_{\text{tot}} = \sum_{l=0}^{L-1} f_\ell = \mathbb{1}^\top \boldsymbol{f}\,, \\ \text{s.t.} \quad & \boldsymbol{P}^*_{N|N} \prec \boldsymbol{\mathcal{V}} \quad \text{and} \quad f_\ell \geq e_{\text{thres}} n_\ell \quad \forall \ell \in [\![0, L-1]\!]\,. \end{aligned} \tag{3.45}$$

Problem (3.45) is quite similar to the one described in Section 3.4.1 and can be solved using the same method as the one used for Problem (3.42), by relying on the KKT conditions. The optimal solution in this case is

$$f_l^* = \begin{cases} e_{\text{thres}} n_\ell, & \text{if } \lambda < \frac{1}{\sum_{b=0}^{n_\ell} 4^b a}\,, \\ \frac{1}{a} \log(\sum_{b=0}^{n_\ell} 4^b a \lambda), & \text{otherwise}\,. \end{cases} \tag{3.46}$$

This solution allows us to compute the optimal energy levels $f_\ell$ for a given energy allocation across the bits. The second step consists of computing the best allocation of bits to each energy group. Given that we only consider small values of $L$, we compute the optimal solution from (3.46) for each possible energy allocation of the bits. Then, the solution with the smallest total energy $\sum_{\ell=0}^{L-1} f_\ell$ is retained. Although Problem 2 leads to a more practical solution, it is expected that the optimal total energy of the memory is higher for Problem 2 than for Problem 1.

## 3.5 Simulation Results

In our simulations, unless explicitly stated, we consider a simple tracking problem where the state vector $\boldsymbol{x}$ is composed of two variables representing the position and velocity of an object. Measurements $y$ only consist of noisy observations of the position of the object. The process matrix $\boldsymbol{F}$ and measurement matrix $\boldsymbol{H}$ are defined as

$$\boldsymbol{F} = \begin{bmatrix} 1 & \delta t \\ 0 & 1 \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{3.47}$$

and the process noise covariance matrix $\boldsymbol{Q}$ and measurement covariance matrix $\boldsymbol{R}$ are given by

$$\boldsymbol{Q} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_x^2 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} \sigma_y^2 \end{bmatrix}. \tag{3.48}$$

where $\delta t = 1$ and $\sigma_x = 0.01$ and $\sigma_y = 10$. The factor $a$ in (3.9) is taken as $a = 12.8$ as in [78]. In all the results presented in this section, the simulations were carried out by replicating the actual bit-flip process following equation (3.9). This section is divided into two parts. We first evaluate the accuracy of the proposed theoretical analysis, and we then provide solutions to the two considered optimization problems.

### 3.5.1 Benefit of taking into account the memory noise in the Kalman filter equations

First, we want to show why it is necessary to modify the Kalman filter equations to take into account the new sources of noise. To demonstrate this, we perform Monte Carlo simulations ($N_{mc} = 10^7$) and measure the covariance matrix of the error on the estimation at step $N = 250$, on a quantized filter ($m = 20$) suffering from memory noise.

Figure 3.1 shows the measured variance of the estimation error on the position and the velocity depending on the energy supplied to the filter. We plot these values for two different filters. In the first one, we use the standard Kalman filter equations such as described in Section 3.2.1. In the second case, we consider a Kalman filter which takes into account the memory noise by using the modified covariance equation proposed in (3.40). As we can see in Figure 3.1, adding a correction term to the filter's equations to take into account the new source of noise lead to a better performance of the filter when the noise level starts to get large. This demonstrates the importance of accurately modeling the sources of errors that can affect the filter.

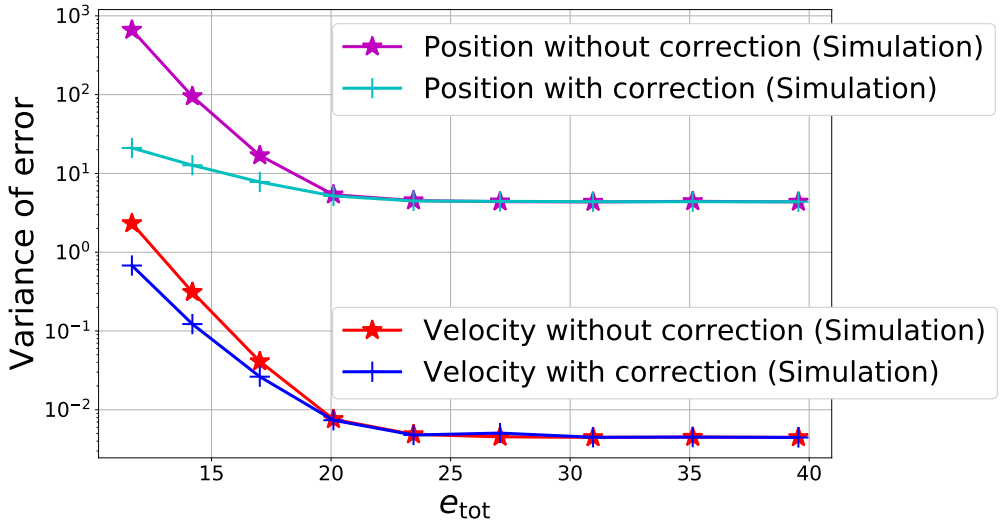



Figure 3.1 Simulated variance of estimation error on the position and velocity depending on the total energy supplied to the memory and if the memory noise is taken into account or not

### 3.5.2 Accuracy of the Theoretical Analysis

To evaluate the accuracy of the proposed theoretical analysis, we perform Monte Carlo simulations ($N_{mc} = 10^7$) and measure the covariance matrix of the error on the estimation at step $N = 250$, thus giving enough time for the filter to converge in normal conditions. This covariance matrix is compared with the theoretical expression of the covariance of the estimation error $\boldsymbol{P}^*_{N|N}$ computed in Section 3.3.4.

Figure 3.2 shows the variance of the estimation errors on the position and the velocity for different values of $m$ in the case of a reliable memory, meaning that we consider only the quantization error and not the memory noise.

We observe that the theoretical predictions of the errors closely match the simulations, which shows the accuracy of our theoretical analysis. From Figure 3.2, we can observe that, for a small number of bits $m$, the quantization error is large and dominates the total estimation error. However, starting from $m = 10$ bits, the estimation error reaches a constant level, which can be interpreted as a minimum bound on the estimation error that one can obtain from a standard Kalman filter for this tracking problem. This shows that, from $m = 10$ bits, the quantization errors become negligible compared to the estimation error achieved by the standard full precision Kalman filter. Thus, at this point, using more bits will not result in minimizing the estimation error, justifying the need for optimizing the parameter $m$.

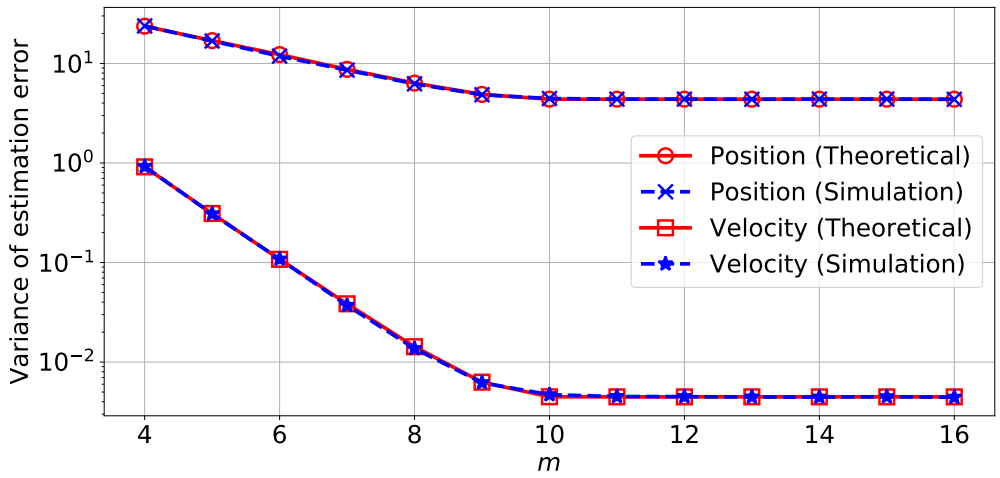



Figure 3.2 Theoretical and simulated variance of estimation error on the position and velocity depending on the numbers of bits in the representation, using a reliable memory

In a second step, we introduce the memory noise in addition to the quantization. Figure 3.3 shows the variance of the estimation error on the position depending on the total energy $e_{\text{tot}}$ for different values of $m$. The variance values were obtained from both Monte Carlo simulations and from the theoretical analysis of Section 3.3.4. Furthermore, Figure 3.4 shows the variance of estimation error on the position depending on the number of bits $m$ for different values of the total energy $e_{\text{tot}}$. The comparison between theoretical results and Monte Carlo simulations show the accuracy of the theoretical analysis that predicts the new computed covariance $\boldsymbol{P}^*_{k|k}$.

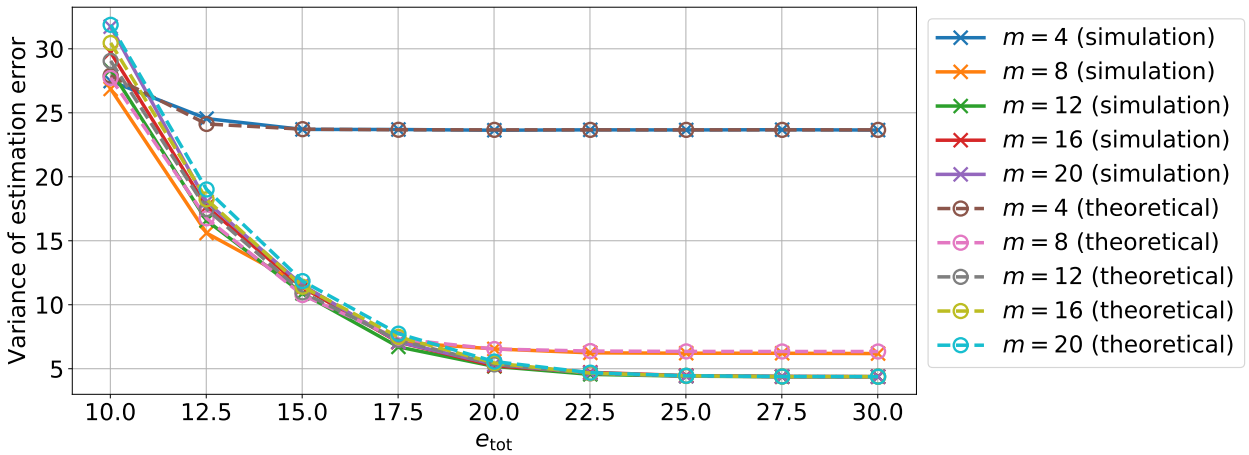



Figure 3.3 Theoretical and simulated variance of estimation error on the position depending on the energy supplied to each variable using an unreliable memory for different numbers of bits in the representation $m$

From Figure 3.3, we can also see that both the number of bits and the total energy can affect the variance of the estimation error. If the number of bits or the energy supplied is too low, then the quantization error or the memory noise will dominate the total estimation error. However, we can see that there is a minimum number of bits, around $m = 12$, from which, given enough energy, it will be possible to reach the minimum possible variance of estimation error.

Moreover from Figure 3.4, we can see that for a low value of supplied energy per variable, the variance of the estimation error will increase with the number of bits as there is too little energy. However, for a larger amount of energy $e_{\text{tot}} > 10$, the variance of the estimation error will decrease with the number of bits since the quantization error decreases. Finally for a large enough number of bits, the total estimation error will only depend on the total energy and on the variance of the estimation error of a reliable full precision filter.

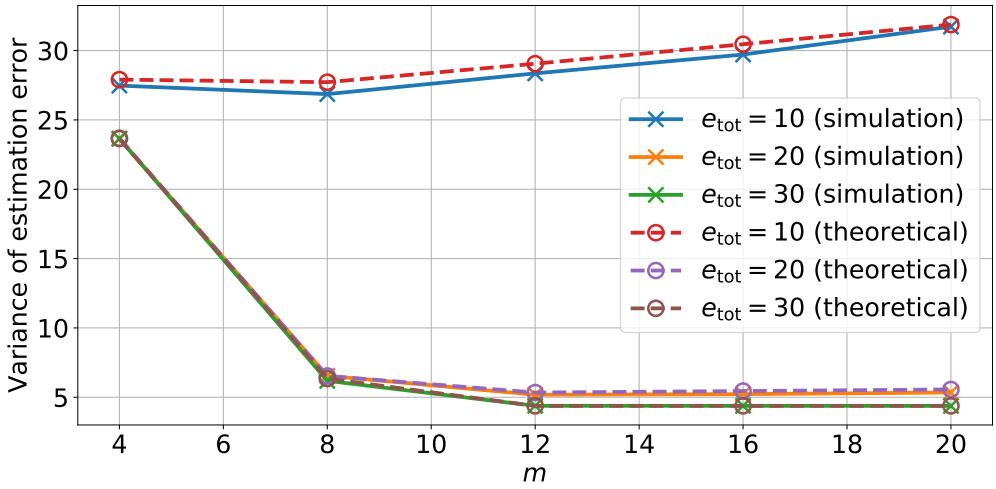



Figure 3.4 Theoretical and simulated variance of estimation error on the position depending on the number of quantization bits using an unreliable memory for different for different total energy values $e_{\text{tot}}$

As the work presented in this chapter was conducted to reduce the energy consumption of the memory of a Kalman filter, it is of a greater utility when the memory is large. For this reason, the results presented before were also tested on a larger Kalman filter with a dimension for the state vector $x$ of $c = 20$. For simulations on this large-size example, we use a state transition model that performs a shifting of the entries of the state to the next state at each

iteration, such as the one used in [133]. That is,

$$\boldsymbol{F}_{i,j} = \begin{cases} 1, & \text{if } i - j = 1\,, \\ 0, & \text{otherwise}\,, \end{cases} \tag{3.49}$$

and $\boldsymbol{F}_{c,1} = 1$. The initial state vector is drawn from a normal distribution.

In this case, the performance of the filter is measured by the trace of the covariance matrix $\boldsymbol{P}_{N|N}$. The results in Figures 3.5 and 3.6 show that the same conclusion can be taken from the simulations done on the small-size Kalman filter and that the method presented in this work can therefore be applied to large-size filters.

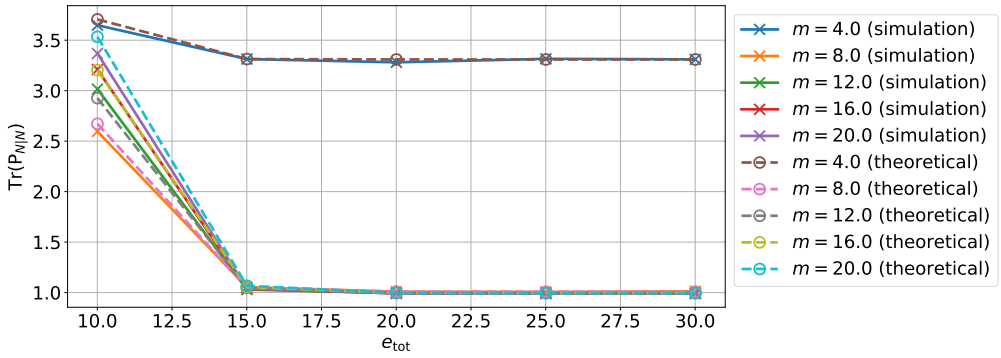



Figure 3.5 Theoretical and simulated variance of estimation error on the position depending on the energy supplied to each variable using an unreliable memory for different numbers of bits in the representation $m$ in the case of the large-size example

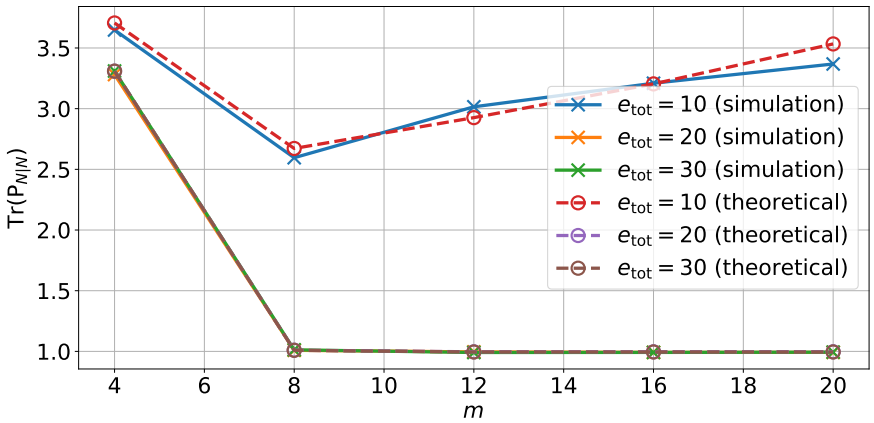



Figure 3.6 Theoretical and simulated variance of estimation error on the position depending on the number of quantization bits using an unreliable memory for different total energy values $e_{\text{tot}}$ in the case of the large-size example

### 3.5.3 Solutions to the Optimization Problems

We now focus on the optimization problems introduced in Section 3.4, starting with the first one. Figure 3.7 shows the amount of energy $e_{\text{tot}}$ needed to store each number in the unreliable memory to achieve a fixed variance of estimation error on the position for each value of $m$.

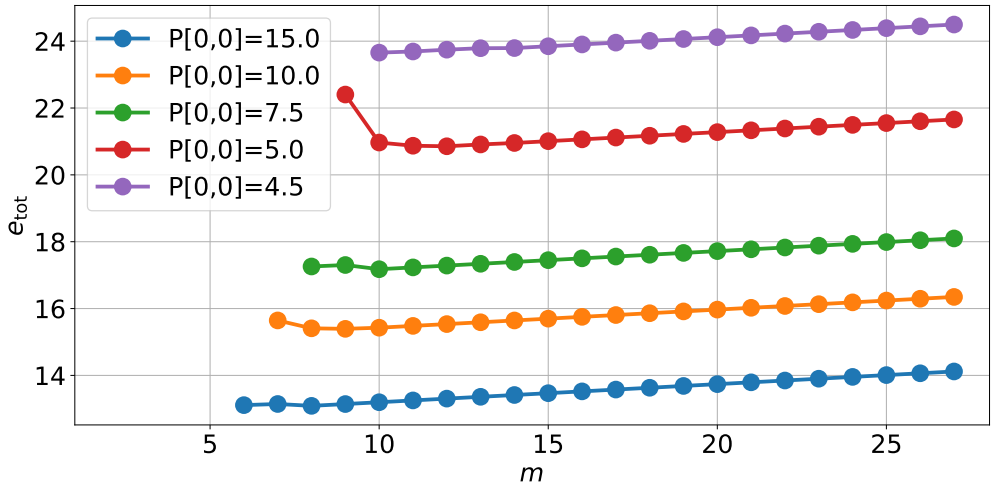



Figure 3.7 The energy needed to store each variable in an unreliable memory to achieve various desired variances of estimation error on the position depending on the number $m$ of bits in the representation with the optimal energy allocation

The total energy $e_{\text{tot}}$ was calculated both using the optimal allocation from Algorithm 1 and using a uniform energy allocation. From Figure 3.8, we can see that the total energy $e_{tot}$ of the memory slightly increases with the number of bits $m + n$. The slight increase in memory consumption is due to the form of the optimal solution (3.43). Indeed, once the minimum number of bits needed to achieve the performance constraint is reached, then additional bits will be set at the minimum energy threshold $e_{\text{thres}}$.

Figure 3.8 compares the optimal solution from Figure 3.7 with a uniform energy allocation. This shows that the optimal energy allocation allows for a significant energy gain compared to the uniform allocation. Here, for the minimum number of bits needed to achieve the performance constraint, the optimal allocation require 56% less energy than the uniform allocation.

We now focus on the second optimization problem defined in Section 3.4.2 at equation (3.45) where only a limited number of energy levels are available. Figure 3.9 shows the total energy needed for each variable in memory to achieve a fixed level of error depending on the number $L$ of energy level possible. For each considered number of levels $L \in [\![1, 7]\!]$, the total energy $e_{\text{tot}}$ was computed for all possible energy allocations using the optimal solution (3.46).

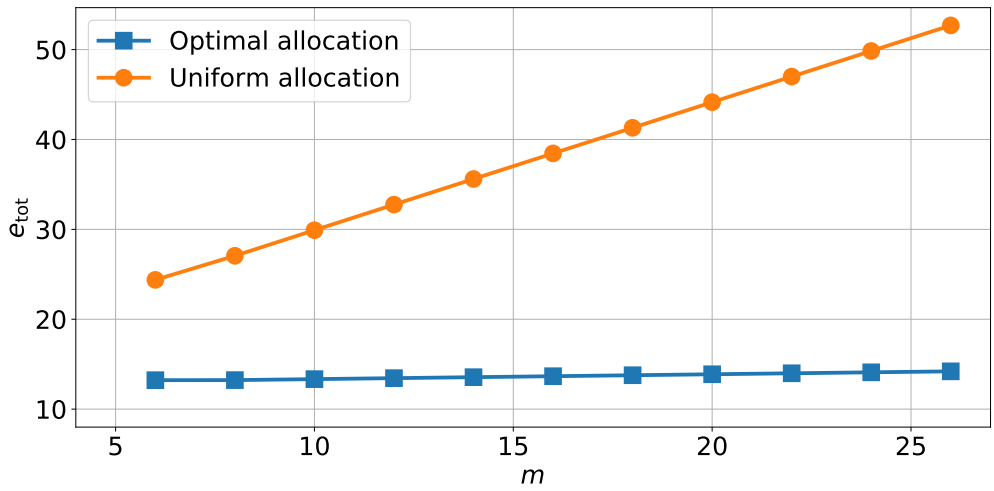



Figure 3.8 Energy needed to store each variable in an unreliable memory to achieve a variance of estimation error on the position $P_{N|N}[0,0] = 15$, depending on the number $m$ of bits in the representation

The minimum energy possible for each value of $L$ was then kept and is shown in Figure 3.9. This minimum energy is compared with the minimum energy needed for Problem (3.41) where there are as many energy levels possible as the number of bits. Here, the total number of quantization bits is $B = 20$.

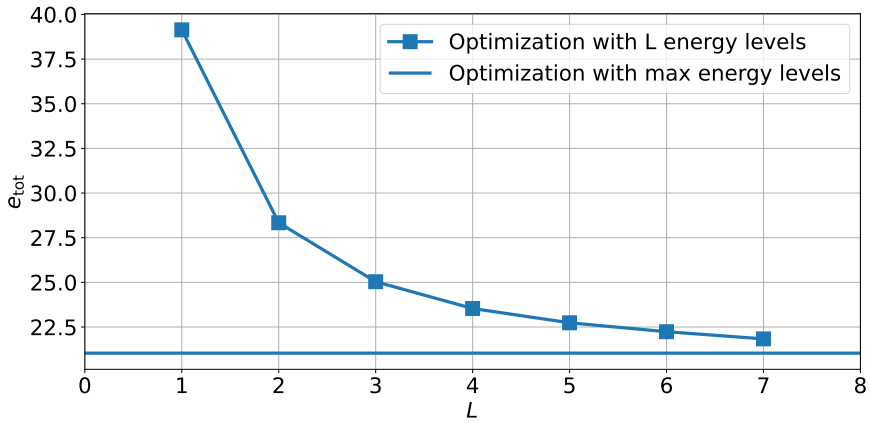



Figure 3.9 Energy needed to store a variable in memory $e_{\text{tot}}$ for different values of energy level available $L$ to achieve a fixed covariance value

We observe that even a small number of energy levels $L$ can lead to significant gains in energy. In this case, only seven levels of energy allow achieving 95% of the maximum energy gain that was obtained in the first optimization problem. When looking at the optimal energy allocation for each value of $L$ bit by bit, we notice that, in most cases, the optimal solution

seems to be when the energy levels are uniformly shared between the bits. This means that, if there are $B$ bits and $L$ levels available and $L$ is a divisor of $B$, then each group of bits assigned to each energy level will have a size of $n_\ell = \frac{B}{L}$.

### 3.6 Conclusions

In this chapter, we studied a quantized Kalman filter implemented with unreliable memories. We provided analytical expressions for the covariance matrix of the estimation error and provided updated filter equations to take into account all considered sources of errors. We proposed and solved two optimization problems that allowed us to find the best trade-offs between the energy consumption and the performance of the filter. The simulation results showed the accuracy of the theoretical analysis and illustrated the significant energy gains provided by our approach. Due to the generic nature of the considered error propagation model, these results could be used for various realistic noise-versus-energy models of unreliable components.

# CHAPTER 4 MEMRISTOR-BASED DEEP NEURAL NETWORKS

## 4.1 Introduction

As mentioned in Chapter 2, memristors [16] are a form of low-power, non-volative memory which allows computing matrix-vector mutiplications (MVM) directly in memory, by programming the memristor conductance values to some specific levels [44, 134]. Hence, memristor-based DNN implementations are able to achieve a significant reduction in energy consumption compared to conventional DNN systems.

However, the programmed conductance levels in memristors may be affected by noise originated from several possible sources [135, 136]. Our literature review has shown than existing methods primarily rely on injecting noise during training, which has also been applied to the weights [96] and activations [137] of memristor-based DNNs.

Even though memristor-based DNN implementations have recently gained in popularity, their analyses, design, and optimization have been primarily empirical in nature [46, 138–140]. In this chapter, we propose an alternative to the vast empirical work on memristors by using a theoretical framework instead. More specifically, our analysis can be used to estimate the performance of a trained DNN deployed on noisy PIM hardware.

As related work, [141] presented a theoretical framework capable of estimating the uncertainty at the output of a network under noisy conditions. Using an extended variational inference framework, their analysis propagates the mean and covariance of the DNN at the output of each layer. The paper clearly identifies the non-linearities of each layer as an issue for moment propagation and proposes two methods: one using first-order Taylor series approximation and another using unscented transformation. While [141] focused on standard implementations suffering from noise or adversarial attacks, moment propagation was also applied to memristor-based implementations, first in [106] which presented a theoretical analysis for studying the influence of conductance variation on a DNN at inference time. However [106] considers a memristor crossbars model for MVM computations based on passive summing circuits, which seems less widely utilized in current experimental implementations compared to the memristor model used in our work. Here, unlike [106, 141], we investigate the case where convolutional layers are directly implemented on memristors, rather than converted to fully-connected layers. This has an important impact on the theoretical analysis based on moment propagation. Moreover, we propose a novel and more accurate method for moment estimation after ReLU activation functions.

In this work, we propose an improved method, in terms of adaptability and precision, for theoretically estimating the influence of noise on the computation of a memristor-based DNN. Particularly, we develop a framework for estimating the MSE between an unreliable neural network implemented using memristors and its reliable counterpart. This framework allows the prediction of the first and second moments of the outputs of a variety of DNN layers. We also present equations for evaluating the power consumption of the memristors used by the network. Finally, we propose an optimization method that allows minimizing the power consumption of a memristor-based DNN to meet a desired MSE. Furthermore, we introduce a runtime-efficient implementation of our theoretical framework and empirically demonstrate a speedup of two orders of magnitude compared to Monte-Carlo simulations. We showcase the correctness of the proposed theoretical analysis when estimating the MSE of a small DNN model applied to a regression task and a larger convolutional neural network (CNN) capable of achieving an accuracy superior to 90% on the CIFAR-10 classification dataset [18]. Additionally, we show that using the theoretical analysis in the previously mentioned optimization problem, it is possible to reach the maximum accuracy of a network with 6% less power consumption than with its non-optimized counterpart. In the end, our results show that our theoretical framework can accurately and efficiently predict the performance of a memristor-based DNN as a function of device characteristics.

The remainder of the chapter is organized as follows: Section 4.2 introduces the memristors and DNN models, Section 4.3 details our theoretical analysis of the memristor-based DNN performance and provides an overview of the computational challenges derived from efficiently implementing our analytical framework, Sections 4.5 and 4.6 describe the proposed optimization method and corresponding results, respectively, and, finally, Section 4.7 provides a summary of our main contributions and future directions.

## 4.2 Models

Before presenting our theoretical analysis, we first need to describe the memristor crossbar model we consider and its associated noise model, as well as how this model is used in the context of MVM on memristors. Then, we introduce the characteristics and notation of the DNNs that will be considered in this paper.

### 4.2.1 Memristor Model

Figure 4.1 illustrates the architecture of the considered memristor crossbar. In accordance with Ohm's Law and Kirchoff's Law, the current in the branch is given by the conductance

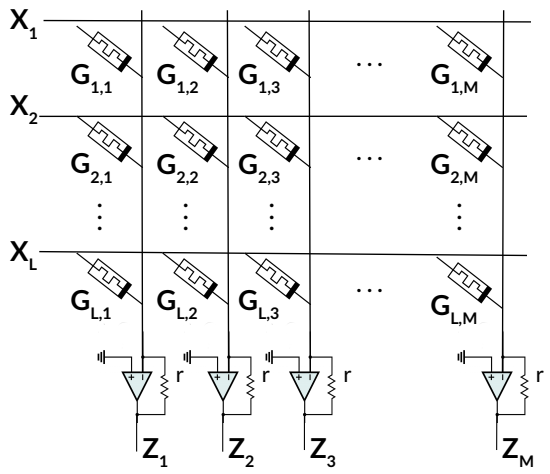



Figure 4.1 Memristor crossbar architecture for MVM

at each node multiplied by the input voltage of the row. These products are then summed along the column. Finally, a transimpedance amplifier (TIA) converts the current into a voltage at the end of each column [142]. The output $z_j$ of the $j$-th column is thus given by

$$z_j = r \sum_{i=1}^{L} g_{i,j} x_i \,, \tag{4.1}$$

where $x_i$ is the voltage at the input of row $i$, $g_{i,j}$ is the conductance of the memristor at row $i$ and column $j$, and $r$ is the feedback resistance of the TIA.

Unfortunately, several practical issues may cause the actual computation to differ from the ideal case presented in equation (4.1). Specifically, conductance values may be affected by fabrication variations and noise during programming [44, 136, 143]. We consider that the memristors conductance ranges from $\mathrm{g_{min}}$ to $\mathrm{g_{max}}$, which respectively represent the minimum and maximum physical conductance values programmable on a memristor. To take variability into account, we represent the programmed conductance values as random variables $\tilde{g}_{i,j}$, which we model as

$$\tilde{g}_{i,j} = g_{i,j} + \epsilon^v_{i,j} \,, \tag{4.2}$$

where $g_{i,j}$ is the desired value and $\epsilon^v_{i,j}$ is a term representing the noise due to memristor variabilities and can be adapted depending on the considered sources of memristor noise such as variability in conductance programming or device-to-device variations. We use $\sigma_v^2$ to denote the variance of $\epsilon^v$. In practice, $\sigma_v^2$ may vary with the conductance value $g_{i,j}$ [136], but for simplicity, we assume here that it is constant. Nevertheless, the analysis proposed in this paper can be easily extended to the case where $\sigma_v^2$ depends on the conductance value.

### 4.2.2 Mapping Dot-Product Computation Into a Memristor Crossbar

Since memristors can only store positive values, we actually use two memristor crossbars for storing one matrix: the $g_{i,j}^{(+)}$ are the positive values of the matrix and the $g_{i,j}^{(-)}$ are the opposite of the negative values. As in [140], the MVM can then be realized by

$$\tilde{z}_j = \sum_{i=1}^{L} r\tilde{g}_{i,j}^{(+)}\tilde{x}_i - \sum_{i=1}^{L} r\tilde{g}_{i,j}^{(-)}\tilde{x}_i \,, \tag{4.3}$$

where $\tilde{z}_j$, $\tilde{g}_{i,j}^{(+)}$, $\tilde{g}_{i,j}^{(-)}$, and $\tilde{x}_i$ are random variables. Moreover, $\tilde{x}_i$ and $\tilde{z}_j$ can be seen as noisy versions of the input $x_i$ and output $z_j$ presented in (4.1), respectively.

Now, an MVM required by a neural network can be formulated as $y = wx$, where $w$ is a weight matrix and $x$ is the input vector. To map this operation into a memristor crossbar, we need to convert the original weight matrix $w$ to two conductance arrays $g^{(+)}$ and $g^{(-)}$. Because of the range $[\mathrm{g_{min}}, \mathrm{g_{max}}]$ of possible conductance values, the weights $w_{i,j}$ are first scaled by a factor $c_{i,j}$, such that

$$w_{i,j}^{s} = c_{i,j} w_{i,j} \,, \tag{4.4}$$

where $c_{i,j} = \frac{\mathrm{g_u} - \mathrm{g_{min}}}{w^u}$, with $w^u$ representing the maximum absolute weight value. In addition, $\mathrm{g_u}$ is chosen depending on the desired trade-off between accuracy and power consumption, and it only needs to be smaller than the maximum physical conductance value $\mathrm{g_{max}}$. Note that the scaling factor $c_{i,j}$ may vary for different parts of the neural network, and in Section 3.4, we will study cases where $c_{i,j}$ can vary from layer to layer, or from column to column.

After computing the scaled weights, we compute the positive and negative memristors values as

$$g_{i,j}^{(+)} = {w_{i,j}^{s}}^{(+)} + \mathrm{g_{min}} \,, \tag{4.5}$$

$$g_{i,j}^{(-)} = {w_{i,j}^{s}}^{(-)} + \mathrm{g_{min}} \,, \tag{4.6}$$

where ${w_{i,j}^{s}}^{(+)} = \lfloor \frac{\mathrm{sgn}(w_{i,j}^{s})+1}{2} \rfloor w_{i,j}^{s}$ and ${w_{i,j}^{s}}^{(-)} = \lfloor \frac{\mathrm{sgn}(w_{i,j}^{s})-1}{2} \rfloor w_{i,j}^{s}$. For simplicity, we define $g_{i,j}$ as $g_{i,j} = g_{i,j}^{(+)} - g_{i,j}^{(-)}$. We note that the random variables $\tilde{g}_{i,j}^{(+)}$ and $\tilde{g}_{i,j}^{(-)}$, which appear in equation (4.3), are the noisy versions of $g_{i,j}^{(+)}$ and $g_{i,j}^{(-)}$, respectively, following the model from (4.2).

### 4.2.3 Computation Models for DNNs and CNNs

We consider a DNN as a sequence of layers, where each layer corresponds to either a MVM or to other types of linear or non-linear transformations. Here, we present the different types of DNN layers that we consider in our analysis, and we explicitly describe their computation models for memristor crossbars. Note that the parts of a DNN that heavily rely on MVMs are fully-connected layers and convolutional layers. Hence, we consider that only these layers are implemented using memristors; other functions in the network such as pooling and non-linear activation functions are assumed to be processed by digital circuits that are not affected by noise, as in [48, 144]. In the following discussions, we consider that the input and output of each layer is a three-dimensional tensor. The input, denoted by $x$, has size $H \times W \times C^i$, while the output, denoted by $z$, has size $E \times F \times C^o$, where $C^i$ and $C^o$ correspond to the number of input and output channels, respectively.

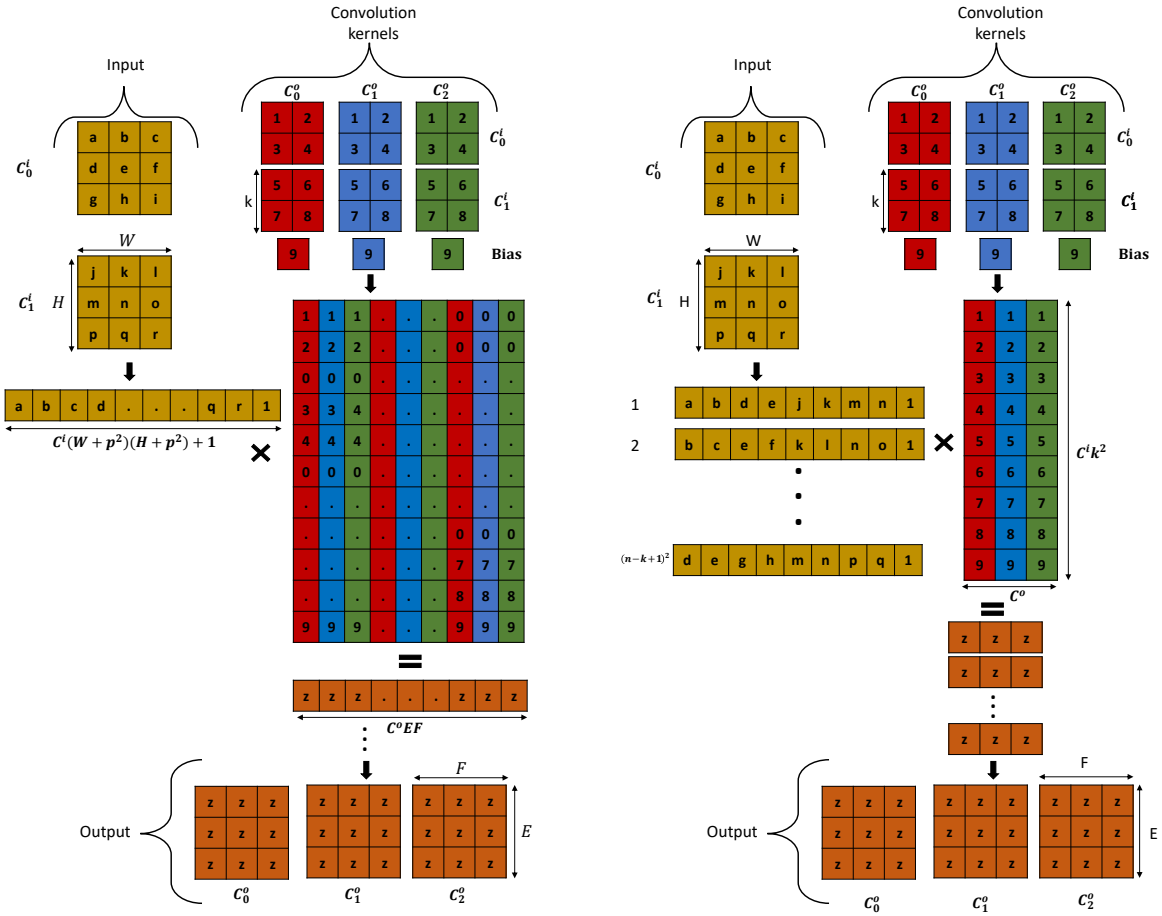
#### Fully-Connected Layer

A fully-connected layer is represented by its weight matrix of size $HWC^i \times EFC^o$. To implement the corresponding dot-product operation using memristors, we construct two memristor crossbars of the same size as the weight matrix plus one additional row for the bias. The conductance values $g_{i,j}^{(+)}$ and $g_{i,j}^{(-)}$ are computed following (4.4) and mapped onto the memristors. Then, the memristor crossbar outputs $\tilde{y_j}^{(+)} = \sum_{i=1}^{L} r\tilde{g}_{i,j}^{(+)}\tilde{x}_i$ and $\tilde{Z_j}^{(-)} = \sum_{i=1}^{L} r\tilde{g}_{i,j}^{(-)}\tilde{x}_i$ are computed. We assume that the difference $\tilde{y_j} = \tilde{y_j}^{(+)} - \tilde{y_j}^{(-)}$, as well as rescaling, $\tilde{z}_j = \frac{\tilde{y_j}}{c}$, are computed by digital circuits outside of the memristor crossbars.

#### Convolutional Layer

We consider a convolutional layer with $C^i$ input channels and $C^o$ output channels, a kernel size $k$, and a padding size $p$. We consider that a convolutional layer may be mapped to a memristor device using one of the following two approaches.

The first approach, which we call *unrolled-linear* (UL), consists in converting the convolutional layer into a fully connected layer. This is illustrated in Figure 4.2a, where the weight matrices of each kernel are unfolded and repeated into a large matrix. Then, the input is flattened into a vector which can be directly multiplied with the unrolled new weight matrix stored using memristors. Note that the weight matrix is therefore of size $C^i(H + p^2)(W + p^2) \times C^o EF$. This approach has the advantage that the computation of each layer can be done in one pass, meaning that only one MVM is performed. On the other hand, it requires storing a large, although sparse, matrix on memristor crossbars.




(a) *Unrolled-linear* mapping (b) *Unfold-repeat* mapping

Figure 4.2 Mapping designs of the convolution operation onto memristors

Moreover, the larger the height of the matrix that is stored on a memristor crossbar, the more noise will be added to the result of the MVM carried out.

In the second approach, which we refer to as *unfold-repeat* (UR), a weight matrix is constructed for all the kernels [96, 145]. Figure 4.2b shows how the mapping of the operation translates to a memristor crossbar. The constructed matrix has size $k^2C^i \times C^o$ with each row containing all the flattened kernels for one output channel. This matrix is then stored on a memristor crossbar. To implement the convolution operation, the input is then unrolled into patches of size $k^2C^i$, and each patch is multiplied with the memristor crossbar. With this approach, a much smaller memristor array is needed, but the number of MVM required is now $(H - k + 1)(W - k + 1)$.

In terms of actual hardware implementation, the second convolution mapping is more practical. This is due to the required size of the memristor crossbar being much smaller, which reduces the amount of noise in the final computation.

### Other linear Operations

We now underline that some non-linear operations such as batch normalization may be implemented by resorting to the previous UL and UR approaches described for convolutional layers. In this case, an equivalent UL or UR can be constructed. Alternatively, a preprocessing fusion step may also be applied, as discussed in Section 4.4. In our theoretical discussions and experimental results, we use average pooling for the pooling operation due to its linear nature (compared to the often-used max-pooling) which facilitates the theoretical analysis.

### Activation Function

As previously mentioned, we consider the DNN non-linear operations to be implemented in digital circuits. Nonetheless, the first and second-order moments must be propagated through the activation function. For most activation functions, evaluating such moments may require an approximation of some operations, which can be computationally expensive and hence reduce the practicality of using a theoretical analysis compared to a Monte-Carlo evaluation of the MSE. Instead, we impose the use of the ReLU activation function and propose an efficient analytical formula to compute the first and second-order moments (see Section 4.3.4).

## 4.3 Theoretical Analysis

To evaluate the robustness to noise of a memristor-based DNN, we compute the MSE between the output of a noisy network implementation and the output of the original network as if implemented on reliable hardware (noise free and perturbation free). Denoting by $z$ the output of the reliable network and by $\tilde{z}$ the output of the noisy variant, we can express the MSE as

$$\mathrm{MSE}[\tilde{z}] = \mathbb{V}[\tilde{z}] + (\mathbb{E}[\tilde{z}] - z)^2 \,. \tag{4.7}$$

As we see from equation (4.7), the computation of the MSE requires two terms for each operation performed on the memristor-enabled computing device: the first and the second-order moments. Therefore, the goal of our theoretical analysis is to compute these moments at the output of each DNN layer.

Since fully-connected and convolutional layers are implemented on memristor crossbars, they are affected by noise, and as such have a significant effect on the successive moments. On the other hand, some operations, namely batch normalization, average pooling, and the activation function, are performed on reliable digital circuits. However, they propagate the moments from earlier layers.

In the following discussions, we denote by $\mu$ and $\gamma$ the mean and covariance matrix of the input $\tilde{x}$ of a given layer, respectively, and we write $\mathbb{V}[\tilde{x}_{c,i,j}] = \gamma^2_{c,i,j}$ and $\mathrm{Cov}[\tilde{x}_{c,i,j}, \tilde{x}_{c',i',j'}] = \gamma^2_{c,i,j,c',i',j'}$. We now present the detailed equations for calculating the mean and covariance of the output of the different layer types considered in this work.

### 4.3.1 Moment Propagation for Fully-Connected Layers

As mentioned in Section 4.2, the noise introduced in each memristor weight is independent from the noise introduced in the other memristors. In other words, the only correlation between outputs is due to the computations performed by previous layers.

The random variable $\tilde{g}_{i,j} = \tilde{g}^{(+)}_{i,j} - \tilde{g}^{(-)}_{i,j}$, has mean $\mathbb{E}[\tilde{g}_{i,j}] = cw_{i,j}$. If $\tilde{g}^{(+)}_{i,j}$ and $\tilde{g}^{(-)}_{i,j}$ have noise variance $\sigma^2$, then $\tilde{g}_{i,j}$ has variance $2\sigma^2$. Hence, the fully-connected layer computations presented in (4.3) followed by rescaling, can be rewritten as

$$\tilde{z}_j = \frac{r}{c}\left(\sum_{i=1}^{L} \tilde{g}_{i,j}\tilde{x}_i + \tilde{B}_j\right) . \tag{4.8}$$

We can then express the first and second moments of $\tilde{z}_j$ as

$$\mathbb{E}[\tilde{z}_j] = r(\sum_{i=1}^{L} w_{i,j}\mu_i + b_j)\,, \tag{4.9}$$

$$\mathbb{V}[\tilde{z}_j] = r^2\left(\frac{2\sigma^2}{c^2} + \sum_{i=1}^{L}\frac{\sigma^2\mu_i^2}{c^2} + \gamma_i^2 w_{i,j}^2 + \frac{\gamma_i^2\sigma^2}{c^2} + \sum_{i=1}^{L}\sum_{i'=1,i'\neq i}^{L} w_{i,j}w_{i'j}\gamma_{i,i'}^2\right), \tag{4.10}$$

and

$$\mathrm{Cov}[\tilde{z}_j, \tilde{z}_{j'}] = r^2\sum_{i=1}^{L}\sum_{i'=1}^{L} w_{i,j}w_{i'j'}\gamma_{i,i'}^2\,. \tag{4.11}$$

### 4.3.2 Moment Propagation for Convolutional Layers

Unlike the fully-connected layer case presented above, all outputs from the same output channel in the UR convolution mapping pass through the same memristors. Hence, the same noise realization is present at different computation stages. To take this into account, we introduce an additional term in the computations of the covariance between two outputs from the same output channel.

We rewrite the computation at the convolution layer followed by the rescaling as

$$\tilde{z}_{c_o,i_o,j_o} = \frac{r}{c_{c_o}}\sum_{c}\sum_{\substack{i=-k/2\\ j=-k/2}}^{k/2}\tilde{g}_{c_o,c,i,j}\tilde{x}_{c,i_o+i,j_o+j} + \tilde{B}_{c_o}\,. \tag{4.12}$$

We then express the analytical expressions of the mean and variance of $\tilde{z}_{c_o,i_o,j_o}$ as

$$\mathbb{E}[\tilde{z}_{c_o,i_o,j_o}] = r\sum_{c}\sum_{\substack{i=-k/2\\ j=-k/2}}^{k/2}\bar{w}_{c_o,c,i,j}\mu_{c,i_o+i,j_o+j} + b_{c_o}\,, \tag{4.13}$$

and

$$\mathbb{V}[\tilde{z}_{c_o,i_o,j_o}] = r^2\left(\sum_{c,c'}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\sum_{\substack{i'=-\frac{k}{2}\\ j'=-\frac{k}{2}}}^{\frac{k}{2}}\varpi_{c_o,c_o,i_o,j_o,i_o,j_o}^{c,c',i,j,i',j'} + \frac{2\sigma^2}{c_{c_o}^2}(1+\sum_{c}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}(\gamma_{c,i_o+i,j_o+j}^2 + \mu_{c,i_o+i,j_o+j}^2))\right), \tag{4.14}$$

respectively, with $\varpi_{c_o,c_o',i_o,j_o,i_o',j_o'}^{c,c',i,j,i',j'} = \bar{w}_{c_o,c,i,j}\bar{w}_{c_o',c',i',j'}\gamma_{c,i_o+i,j_o+j,c',i_o'+i',j_o'+j'}^2$.

For the covariance calculation, since several inputs will encounter the same noise realization through the same memristor, we analyze two cases covering the computation of the covariance : in the first case, two outputs are from the same output channel, and in the second case,

they are from different output channels. In the first case, *i.e.* if $c_o = c'_o$, we have

$$\mathrm{Cov}[\tilde{z}_{c_o,i_o,j_o}, \tilde{z}_{c_o,i'_o,j'_o}] = r^2\Big(\sum_{c,c'}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\sum_{\substack{i'=-\frac{k}{2}\\ j'=-\frac{k}{2}}}^{\frac{k}{2}} \varpi^{c,c',i,j,i',j'}_{c_o,c'_o,i_o,j_o,i'_o,j'_o} + \frac{2\sigma^2}{c^2_{c^o}}\Big(1 + \sum_{c}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}} \gamma^2_{c,i_o+i,j_o+j,c,i'_o+i,j'_o+j} + \mu_{c,i_o+i,j_o+j}\mu_{c,i'_o+i,j'_o+j}\Big)\Big)\,. \quad (4.15)$$

Otherwise, the covariance between two outputs of different output channels ($c_o \neq c'_o$) is computed as

$$\mathrm{Cov}[\tilde{z}_{c_o,i_o,j_o}, \tilde{z}_{c'_o,i'_o,j'_o}] = r^2\sum_{c,c'}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\sum_{\substack{i'=-\frac{k}{2}\\ j'=-\frac{k}{2}}}^{\frac{k}{2}} \varpi^{c,c',i,j,i',j'}_{c^o,c'_o,i^o,j^o,i'_o,j'_o}\,. \quad (4.16)$$

### 4.3.3 Moment Propagation for Average Pooling Layers

We consider a 2D average pooling layer with stride $s$. The operation done at this layer is

$$\tilde{z}_{c,i,j} = \frac{1}{s^2}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s}\tilde{x}_{c,k,l}\,. \quad (4.17)$$

Since the average pooling layer is a linear operation, its moments can be computed exactly as:

$$\mathbb{E}[\tilde{z}_{c,i,j}] = \frac{1}{s^2}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s}\mu_{c,k,l}\,, \quad (4.18)$$

$$\mathbb{V}[\tilde{z}_{c,i,j}] = \frac{1}{s^4}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s}\sum_{m=i}^{i+s}\sum_{n=j}^{j+s}\gamma^2_{c,k,l,c,m,n}\,, \quad (4.19)$$

and

$$\mathrm{Cov}[\tilde{z}_{c,i,j}, \tilde{z}_{c',i',j'}] = \frac{1}{s^4}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s}\sum_{m=i'}^{i'+s}\sum_{n=j'}^{j'+s}\gamma^2_{c,k,l,c',m,n}\,. \quad (4.20)$$

### 4.3.4 Moment Propagation for the Activation Functions

Activation functions introduce non-linearities in the network. Letting $f$ denote the activation function, the operation done at this layer can be written as $\tilde{z}_{i,j} = f(\tilde{x}_{i,j})$. Because of the

non-linear nature of $f$, it can be complex to compute exactly the propagation of the first and second-order moments through the activation functions. This step is thus the only one in the proposed framework where some approximations are used.

### Taylor Approximation

A first possibility to approximate the moments after a non-linear activation function $f$ is via Taylor expansions [106]:

$$\mathbb{E}[f(\tilde{z}_{i,j})] \approx f(\mu_j) + \frac{1}{2} f''(\mu_{i,j}) \gamma_{i,j}^2 , \tag{4.21}$$

$$\mathbb{V}[f(\tilde{z}_{i,j})] \approx \frac{1}{2} g''(\mu_j) \gamma_{i,j}^2 - f(\mu_{i,j}) f''(\mu_{i,j}) \gamma_{i,j}^2 , \tag{4.22}$$

$$\mathrm{Cov}[f(\tilde{z}_{i,j}), f(\tilde{z}_{i',j'})] \approx f'(\mu_{i,j}) f'(\mu_{i',j'}) \gamma_{i,j,i',j'}^2 , \tag{4.23}$$

where $g = f^2$.

In the case of a ReLU activation function, the second order derivative is zero and therefore the mean and variance can be written as:

$$\mathbb{E}[f(\tilde{z}_{i,j})] \approx f(\mu_{i,j}) , \tag{4.24}$$

$$\mathbb{V}[f(\tilde{z}_{i,j})] \approx f'(\mu_{i,j}) \gamma_{i,j}^2 , \tag{4.25}$$

with $f(\mu) = \begin{cases} 0 & \mu \leq 0 \\ \mu & \mu > 0 \end{cases}$ and $f'(\mu) = \begin{cases} 0 & \mu \leq 0 \\ 1 & \mu > 0 \end{cases}$.

### Gaussian Approximation

In this work we propose to use another method to compute the mean and variance of the output activations. This requires to know the probability distribution of the inputs of the function. Since we cannot know the exact distribution, we assume the output of fully-connected and convolutional layers to follow a normal distribution. This assumption is supported by the fact that in the memristor computations, only the inputs are dependent, but not the noise added on each memristor. Therefore, we can apply the central limit theorem. The validity of this hypothesis was also empirically verified through simulations presented in Section 4.6.

We can then express the mean and variance of the output of the activation function $f$ as

$$E[\tilde{z}_{i,j}] = \int_{-\infty}^{\infty} f(x) \frac{1}{\gamma_{i,j}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x - \mu_{i,j}}{\gamma_{i,j}}\right)^2\right) dx \tag{4.26}$$

and

$$\mathbb{V}[\tilde{z}_{i,j}] = E[\tilde{z}_{i,j}^2] - E[\tilde{z}_{i,j}]^2 = \int_{-\infty}^{\infty} f(x)^2 \frac{1}{\gamma_{i,j}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_{i,j}}{\gamma_{i,j}}\right)^2\right) dx - E[\tilde{z}_{i,j}]^2 \,. \quad (4.27)$$

In the case of the ReLU activation function considered in this thesis, we show that these integrals have closed-form expressions given by

$$\mathbb{E}[\tilde{z}_{i,j}] = \frac{\gamma_{i,j}}{\sqrt{2\pi}} e^{-\frac{\mu_{i,j}^2}{2\gamma_{i,j}^2}} + \frac{\mu_{i,j}}{2}\left(1 - \operatorname{erf}\left(\frac{-\mu_{i,j}}{\gamma_{i,j}\sqrt{2}}\right)\right) \quad (4.28)$$

and

$$\mathbb{V}[\tilde{z}_{i,j}] = \left(\frac{\mu_{i,j}^2}{2} + \frac{\gamma_{i,j}^2}{2}\right)\left(1 - \operatorname{erf}\left(\frac{-\mu_{i,j}}{\gamma_{i,j}\sqrt{2}}\right)\right) + \frac{\gamma_{i,j}\mu_{i,j}}{\sqrt{2\pi}} e^{-\frac{\mu_{i,j}^2}{2\gamma_{i,j}^2}} - \left(\mathbb{E}[\tilde{z}_{i,j}]\right)^2 \,, \quad (4.29)$$

where erf is the Gauss error function.

*Proof.* In the case of the ReLU function we need to compute the integrals

$$\Psi_1 = \int_0^{\infty} x \frac{1}{\gamma_{i,j}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_{i,j}}{\gamma_{i,j}}\right)^2\right) dx \,, \quad (4.30)$$

and

$$\Psi_2 = \int_0^{\infty} x^2 \frac{1}{\gamma_{i,j}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_{i,j}}{\gamma_{i,j}}\right)^2\right) dx \,. \quad (4.31)$$

We write $\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$ and $\Phi = \frac{1}{2}(1 + \operatorname{erf}(\frac{x}{\sqrt{2}})$ .

From [146],

$$F_1(x) = \int x \frac{1}{\gamma}\phi(\frac{x-\mu}{\gamma})dx = -\gamma\phi(\frac{x-\mu}{\gamma}) + \mu\Phi(\frac{x-\mu}{\gamma}) + C \,. \quad (4.32)$$

We can then compute the finite integral $\Psi_1$ as

$$\Psi_1 = \lim_{x\to\infty} F_1(x) - F_1(0) = \mu + \gamma\phi(\frac{\mu}{\gamma}) - \mu\Phi(\frac{\mu}{\gamma}) \quad (4.33)$$

$$= \gamma\phi(-\frac{\mu}{\gamma}) + \mu(1 - \phi(-\frac{\mu}{\gamma})) \,. \quad (4.34)$$

Also from [146]:

$$F_2(x) = \int x^2 \frac{1}{\gamma}\phi(\frac{x-\mu}{\gamma})dx = \gamma(-x-\mu)\phi(\frac{x-\mu}{\gamma}) + (\mu^2+\gamma^2)\Phi(\frac{x-\mu}{\gamma}) + C. \tag{4.35}$$

We can then compute the finite integral $\Psi_2$ as

$$\Psi_2 = \lim_{x\to\infty} F_2(x) - F_2(0) = (\mu^2+\gamma^2) + \gamma\mu\phi(\frac{\mu}{\gamma}) - (\mu^2+\gamma^2)\Phi(\frac{-\mu}{\gamma}) \tag{4.36}$$

$$= \gamma\mu\phi(\frac{\mu}{\gamma}) + (\mu^2+\gamma^2)(1-\Phi(\frac{-\mu}{\gamma}))\,. \tag{4.37}$$

□

The covariance can still be computed through the Taylor expansion with equation (4.23).

### 4.3.5 Power Consumption

We now use the previous expressions of the moments to compute the mean power usage of each memristor crossbar for the inference of one input. We separate the power usage of the memristor crossbars into two parts: the power usage of the memristors and the power usage of the TIAs.

#### Fully-Connected Layer and UL Convolution

We first derive an estimation of the power consumption of the memristor computations when using a fully-connected layer, which also directly extends to our UL convolution mapping. For the power consumption, it is possible to express directly the power usage for a pair of memristors storing the positive and negative weight of the same position in the original weight matrix. Defining $P_{i,j}^{(\text{mem})} = P_{i,j}^{(\text{mem})(+)} + P_{i,j}^{(\text{mem})(-)}$, the power consumption of each pair of memristor can be written as $P_{i,j}^{(\text{mem})} = |\tilde{g}_{i,j}|\,\tilde{x}_i^2$ . We then compute the mean $\mathbb{E}[\tilde{g}_{i,j}\tilde{x}_i^2]$ as

$$\mathbb{E}[P_{i,j}^{(\text{mem})}] = (c\,|w_{i,j}| + 2\,\text{g}_{\min})(\gamma_{i,j}^2 + x_{i,j}^2) \tag{4.38}$$

$$\mathbb{E}[P_{\text{bias},j}^{(\text{mem})}] = c\,|w_{\text{bias},j}| + 2\,\text{g}_{\min}\ . \tag{4.39}$$

Moreover, the power consumption of each TIA is evaluated for both the positive and negative memristor arrays by

$$P_j^{(\mathrm{TIA})^{(+)}} = r(\sum_{i=1}^{L} \tilde{g}_{i,j}^{(+)} \tilde{x}_i)^2 = \frac{\tilde{z}_j^{(+)^2}}{r} \tag{4.40}$$

$$P_j^{(\mathrm{TIA})^{(-)}} = r(\sum_{i=1}^{L} \tilde{g}_{i,j}^{(-)} \tilde{x}_i)^2 = \frac{\tilde{z}_j^{(-)^2}}{r}\,, \tag{4.41}$$

with

$$\mathbb{E}[P_j^{(\mathrm{TIA})^{(+)}}] = \frac{\mathbb{E}[\tilde{z}_j^{(+)}]^2 + \mathbb{V}[\tilde{z}_j^{(+)}]}{r} \tag{4.42}$$

$$\mathbb{E}[P_j^{(\mathrm{TIA})^{(-)}}] = \frac{\mathbb{E}[\tilde{z}_j^{(-)}]^2 + \mathbb{V}[\tilde{z}_j^{(-)}]}{r}\,. \tag{4.43}$$

Hence, the mean power consumption of each layer is

$$\mathbb{E}[P_{\mathrm{tot}}] = \sum_{j=1}^{L} \left( \sum_{i=1}^{L} \mathbb{E}[P_{i,j}^{(\mathrm{mem})}] + \mathbb{E}[P_j^{(\mathrm{TIA})^{(+)}}] + \mathbb{E}[P_j^{(\mathrm{TIA})^{(-)}}] \right). \tag{4.44}$$

**UR Convolution**

We consider a convolutional layer with kernel size $k$, $C^i$ input channels and $C^o$ output channels and $H \times W$ the height and width of each feature map. Similarly to the fully-connected case, we can express the power consumption of each memristor during inference except that, in this case, we have to sum it by the number of times a MVM product needs to be done to complete the computation for one input map:

$$P_{c_o,c_i,i,j}^{(\mathrm{mem})} = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} |\tilde{g}_{c_o,c_i,i,j}|\, \tilde{x}_{c_i,m,n}^2\,. \tag{4.45}$$

We then compute the mean of this power for each weight stored on a memristor as

$$\mathbb{E}[P_{c_o,c_i,i,j}^{(\mathrm{mem})}] = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} \mathbb{E}[|\tilde{g}_{c_o,c_i,i,j}|\, \tilde{x}_{c_i,m,n}^2] \tag{4.46}$$

$$= \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} (c_{c_o}\, |w_{c_o,c_i,i,j}| + 2\, \mathrm{g_{min}})(\gamma_{c_i,m,n}^2 + x_{c_i,m,n}^2) \tag{4.47}$$

and

$$\mathbb{E}[P^{(\text{mem})}_{\text{bias},c_o}] = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} c_{c_o} \, |w_{\text{bias},c_o}| \tag{4.48}$$

$$= (M-k+1)^2 (c_{c_o} \, |w_{\text{bias},c_o}| + 2\, \mathrm{g}_{\min}) \tag{4.49}$$

for each bias stored on a memristor.

For the power consumption of each TIA, if $\tilde{z}_{c^o,m,n}$ is the result of the convolution, we have

$$P^{\text{TIA}(+)}_{c_o} = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} \frac{\tilde{z}^{(+)^2}_{c^o,m,n}}{r} \,, \tag{4.50}$$

$$\mathbb{E}[P^{\text{TIA}^{(+)}}_{c_o}] = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} \frac{\mathbb{V}[\tilde{z}^{(+)}_{c^o,m,n}] + \mathbb{E}[\tilde{z}^{(+)}_{c^o,m,n}]^2}{r} \,, \tag{4.51}$$

and

$$\mathbb{E}[P^{\text{TIA}^{(-)}}_{c_o}] = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} \frac{\mathbb{V}[\tilde{z}^{(-)}_{c^o,m,n}] + \mathbb{E}[\tilde{z}^{(-)}_{c^o,m,n}]^2}{r} \,. \tag{4.52}$$

We can then express the mean of the total power consumption of one UR convolutional layer as

$$\mathbb{E}[P_{\text{tot}}] = \sum_{c_o=1}^{o} \left( \mathbb{E}[P^{(\text{TIA})^{(+)}}_{c_o}] + \mathbb{E}[P^{(\text{TIA})^{(-)}}_{c_o}] + \mathbb{E}[P^{(\text{mem})}_{\text{bias},c_o}] + \sum_{i=1}^{k} \sum_{j=1}^{k} \sum_{c_i=1}^{d} \mathbb{E}[P^{(\text{mem})}_{c_o,c_i,i,j}] \right) . \tag{4.53}$$

## 4.4 Implementation Details

A runtime-efficient implementation of our theoretical analysis is essential to ensure the practicality of our approach compared to Monte-Carlo simulations. Note that computing the theoretical equations for a given DNN and input tensor $x$ is similar to computing its outputs, i.e. presenting an input to the DNN and forward propagating the output of each layer to the next; with the exception that the inputs and outputs consists of the triplet $\mu$, $\gamma$ and $P$.

There are a few efficiency issues that can be encountered if trying to implement our framework naively which can lead to a heavily memory-bound implementation, thus limiting the size of input batches and preventing the use of larger DNNs, datasets, and optimization techniques for the memristor parameters. Namely, the covariance tensor $\gamma$ tends to be very large for the first layers of the DNN since it grows quadratically in the number of output activations of the layer. This may result in out-of-memory errors early in the computations. Since it is

initialized with zeros, to lessen the memory burden, $\gamma$ is represented only by its size and is lazily instantiated when needed, delaying the memory allocation to at least the second layer which may have a reduced size. This approach simplifies computations in the first layer. Additionally, operator fusion is a technique that can be used in deep neural network deployment to merge sequential operations [147, 148], such as batch normalization and convolution, to increase computational efficiency and reduce memory footprint. In our case, we apply this operator fusion to the network prior to the theoretical analysis in our experiments.

## 4.5 Optimization

The maximal programming conductance value $\mathrm{g_u}$ has a direct impact on both the power consumption of the device and the MSE at the output of the DNN. To address this trade-off, we consider a DNN represented by a function $f(.)$, with a target input $x$ and a set of parameters $W$ that includes the weights and biases of the network. We consider that $\mathrm{g_u}$ belongs to the interval $[\mathrm{g_{min}}, \mathrm{g_{max}}]$, where possible $\mathrm{g_u}$ values correspond to different trade-offs between power consumption and MSE. While lower values lead to smaller power requirements but increased MSE, higher values have the opposite effect. Based on this observation, we formulate the following optimization problem:

$$\begin{aligned} \min_{\mathrm{g_u}} \quad & \mathbb{E}(P_{tot}) \\ \text{s.t.} \quad & \mathbb{MSE}(f(x, W, \mathrm{g_u})) \leq \nu\,, \\ & \mathrm{g_{min}} < \mathrm{g_u} \leq \mathrm{g_{max}} \end{aligned} \tag{4.54}$$

where $\nu$ represents the target MSE to be achieved by the network. This optimization problem represents finding the best values for $\mathrm{g_u}$ which will minimize the power usage of the memristors for a desired MSE constraint.

Although the optimization problem (4.54) aims to satisfy a constraint on the MSE, this metric is not always the end goal in machine learning problems. For example, in the case of a classification problem, it is often the accuracy of the network that needs to be maximized. However, as will be shown in Section 4.6, the MSE can be considered as a good proxy for these other metrics, in particular for the accuracy in a classification task.

In what follows, we address the optimization problem (4.54) for three cases, called designs, of increasing complexity with respect to $\mathrm{g_u}$. Each design increases the degrees of freedom of the optimization compared to the previous one, so as to ideally achieve larger power gains:

1. In the *scalar design*, we consider only one value of $g_u$ for the whole DNN. To do so, we compute $w^u$ as the maximum absolute weight value of all the network weights. Therefore, we compute and apply the same scaling factor for all the weight matrices of the network.

2. In the *layer-wise design*, a distinct $g_u$ is used for each fully connected or convolution layer in the DNN. In this case, we compute one $w^u$ for each layer as the maximum absolute weight of a given layer. Hence, a different scaling factor is computed for each layer depending on $w^u$ and the chosen value of $g_u$.

3. Finally, in the *column-wise design*, we have a vector of $g_u$ for each fully connected or convolution layer, which have the size of the number of outputs of the respective layers. For this, we consider a different value of $w^u$ for each column of each weight matrix. A different scaling factor is then applied to each column of each memristor crossbar to give more flexibility for reducing the power usage of the memristors.

Note that each of the following designs is also associated with a different methodology to compute $w_{max}$.

Due to the forms of the equations for computing the MSE and the power usage, no analytical way of solving this optimization problem could be proposed. Indeed, the problem is non convex, and also the results of a layer affect the following layers so it is not possible to compute them separately. With these issues in mind, to solve problem (4.54) related to each design, we propose to use a heuristic optimizing search, based on a genetic algorithm [149]. Genetic algorithms have the advantage of more easily avoiding local optima and they can be used with a fitness function of any form. Note that depending on the considered design for $g_u$, the number of parameters to optimize may vary, which directly influences the number of iterations of the genetic algorithm required to converge toward a good solution. To reduce the optimization time, we use the solution found in the case of a scalar $g_u$ to initialize the optimization problem in the case of a layer-wise $g_u$. Similarly, we use the solution of the layer-wise $g_u$ to initialize the optimization of the column-wise $g_u$.

## 4.6 Experimental Results

This section presents numerical results comparing our theoretical analysis to Monte-Carlo simulations for several types of DNNs. It also provides optimization results for the three considered designs. In addition, it provides a study of the time efficiency of our implementation of the theoretical framework described in this paper.

### 4.6.1 Experimental Setups

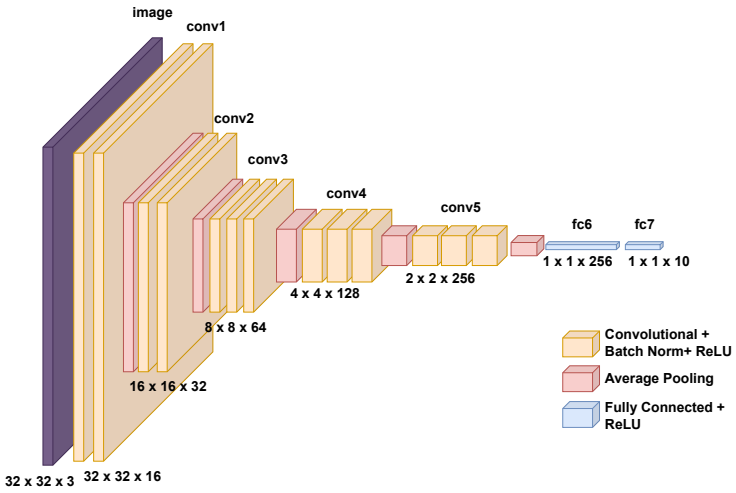



Figure 4.3 Architecture of the classification network used in our experiments

To assess the accuracy of our theoretical analysis and to evaluate our optimization process, we used two different neural networks trained on different tasks: we trained a small DNN on a regression problem, and a larger DNN on a classification problem. Our goal is to showcase the ability of our analytical framework to efficiently scale to larger DNNs while still maintaining its accuracy.

For the regression problem, we used the naval dataset [150] and trained a simple feed-forward network with a hidden layer of 50 neurons and a ReLU activation function for 200 epochs using the Adam optimizer [151] with a learning rate of 0.01. For the classification problem, we trained a convolutional neural network on the CIFAR-10 dataset [18]. The DNN architecture is presented in Figure 4.3 and consists of a series of convolutional layers with a kernel size of 3, a unit stride, and a varying number of filters: 16, 32, 64, 128, and 256 filters were considered. An average pooling layer follows each series and the final part of the network is made of a classifier module composed of two fully-connected layers. We use the ReLU activation function, and dropout is applied after each average pooling during training. The network was trained for 200 epochs using stochastic gradient descent (SGD) with a momentum of 0.9, a weight decay of $5 \times 10^{-4}$, and an initial learning rate of 0.1 (reduced by a factor of 10 after every 50 epochs). Unless stated otherwise, the framework uses the gaussian approximation to compute the activation function.

### 4.6.2 Accuracy of the MSE Estimation

#### Regression Problem

Figure 4.4(a) shows the MSE between the output of the network implemented using memristors and the output of the same network implemented on a reliable system. The MSE is shown as a function of the conductance noise variance $\sigma$, while each curve corresponds to different values of $g_u$. The results were computed using the theoretical analysis as well as using Monte-Carlo simulations. Similarly, Figure 4.4(b) shows the MSE between the memristor network and the ground truth of the data, which is a standard method for measuring the performance of a neural network for a regression problem. In both cases, we observe that the simulations and the analytical solution match almost perfectly. Moreover, we see that as the variance of the memristor noise decreases, the MSE also decreases and converges toward the minimum MSE achievable with a reliable network. This confirms this intuition that as $g_u$ increases, the network is able to tolerate a higher level of noise for the same performance.

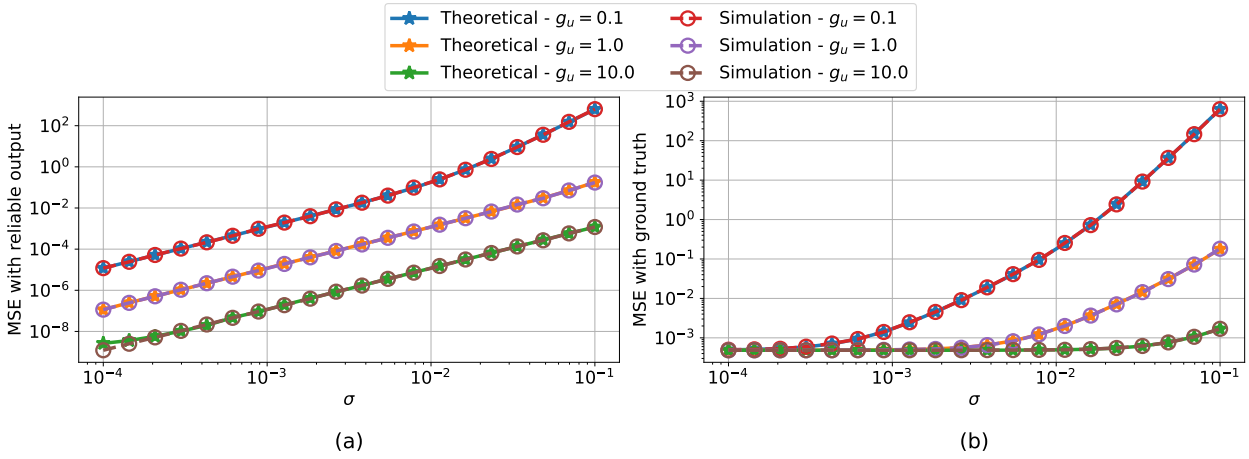



Figure 4.4 Theoretical and Monte-Carlo computations of the MSE depending on the noise variance $\sigma$ for different values of $g_u$

#### Classification Problem

We next studied if the analytical formulas match the simulations for the considered classification DNN. Figure 4.5 shows the MSE between the output of the memristor network and a reliable network as a function of the noise variance $\sigma$. We plot the results for two implementations, one using the unfolded convolution mapping (UR), and one using the unrolled convolution mapping (UL). In all cases, $g_u$ is chosen such that the scaling factor $c$ equals 1. We observe that the simulation almost completely matches with the theoretical results.

On the right y-axis, we also show the accuracy of the noisy classification CNN depending on $\sigma$. We observe the ability of the network to tolerate some error in its computation in the low-noise regime while still maintaining its original accuracy. As mentioned in Section 4.2.3, with the UR convolution the amount of noise on the computation is lower than with the UL convolution when considering the same memristor noise variance $\sigma$. Hence, the MSE is lower for the UR convolution, which transposes to a better capacity for maintaining a high accuracy. Note that the link between MSE and classification accuracy is discussed further in Section 4.6.4.

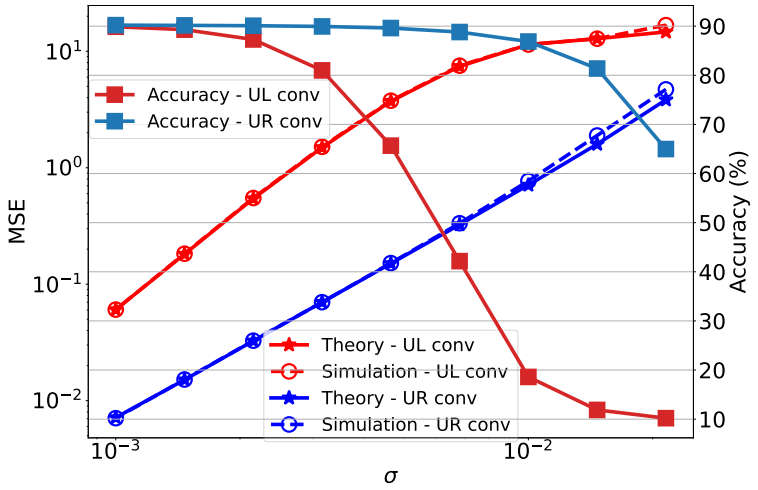



Figure 4.5 MSE and accuracy comparison between the noisy and original DNNs for a classification problem depending on $\sigma$ for the *Unrolled-Linear* convolution mapping (UL conv) and the *Unfold-Repeat* convolution mapping (UR conv)

Figure 4.6 also shows the MSE of the output of the network as a function of the noise variance $\sigma$. The framework was implemented using either the Taylor approxmation or the Gaussian approximation for estimating the moments at the output of the activation function. By comparing the MSE in both of these cases to the one computed with Monte-Carlo simulations, we can see that when the noise level is too large, the estimation computed using Taylor expansions start to diverge from the correct result. On the other hand, the framework using our proposed method still closely match the MSE at all levels of noise.

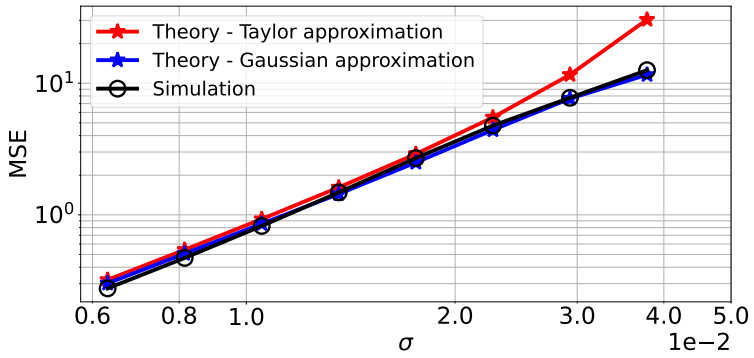



Figure 4.6 MSE comparison between using Taylor expansions and our Gaussian approximation for the theoretical framework depending on $\sigma$

### 4.6.3 Run Time Comparison With Monte-Carlo Simulations

Among other features, the theoretical analysis presented in Section 4.3 allows for evaluating the MSE of an unreliable network faster than using Monte-Carlo simulations. To quantify the execution time reduction compared to Monte-Carlo simulations, we first estimate the number of Monte-Carlo iterations needed using the following equation [152]:

$$n = \left[ \frac{z_{\alpha/2}\overline{s}}{p\overline{x}} \right]^2 . \tag{4.55}$$

In this expression, $z_{\alpha/2}$ is a parameter related to the desired confidence interval, and it can be retrieved from tables of the cumulative distribution function for a normally distributed random variable [152], $p$ is the percentage by which we allow the result to differ from the true value, and $\overline{s}$ and $\overline{x}$ are the sample standard deviation and sample mean, respectively. The values of $\overline{s}$ and $\overline{x}$ may be computed using the Monte-Carlo simulations through a substantial number of iterations to grasp an accurate estimate. Equation (4.55) allows us to compute the minimum number of iterations needed to compute an MSE which will fall within a specific interval from the true value for a certain confidence level. For example, using $z_{\alpha/2} = 1.96$ and $p = 0.01$ translates to a 95% confidence that the computed MSE will not be above or below a 1% difference from the true MSE.

Figure 4.7 shows a comparison between the run time needed for computing the MSE of an input from the theoretical analysis and from Monte-Carlo simulations on the classification network.

The run time is measured for both types of convolution mappings. For the simulations, we measure the Monte-Carlo run time for two possible confidence intervals (which directly

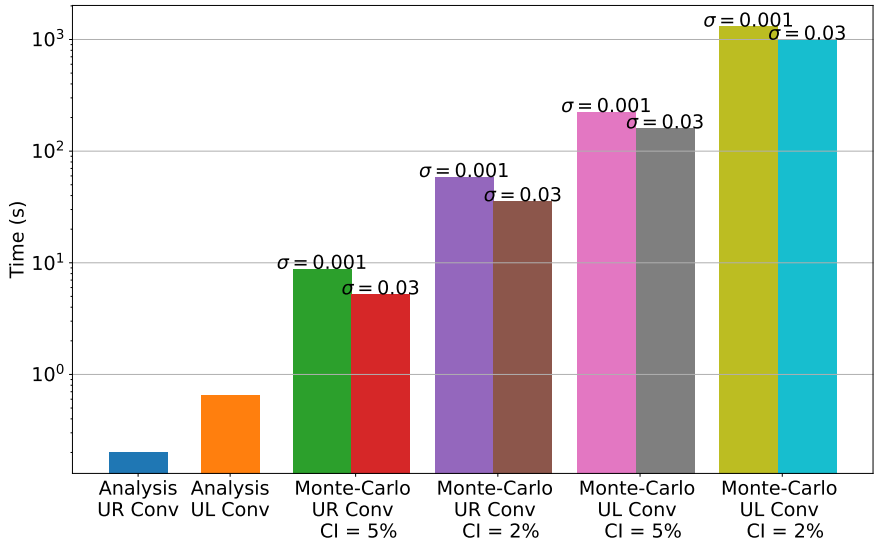



Figure 4.7 Run time comparison between the theoretical analysis implementation and Monte-Carlo simulations

influence the number of Monte-Carlo iterations) and two $\sigma$ values. The computations were done using one A100 Nvidia GPU and an AMD Milan 7413 CPU. The run time measurements for each data point were repeated 10 times and the results were then averaged. We observe that, for the unfolded-repeat convolution mapping, the theoretical analysis is between 26 to 290 times faster than the Monte-Carlo simulations. The same trend is observed for the unrolled-linear mapping, where the acceleration factor is between 243 to 1982 depending on the desired confidence interval.

### 4.6.4 Optimization

#### Regression Problem

We investigate the three designs introduced in Section 3.4. In each case, we want to minimize the power usage under a specified constraint of the MSE between the output and the ground truth. The optimization is done using a genetic algorithm [149] with a population of size 100, a batch of 128 samples, and noise variance $\sigma^2 = 0.01$. The results are presented in Figure 4.8. We observe that layer-wise optimization achieves the same performance as the scalar case for less power usage. However, going from the layer-wise design to the column-wise does not seem to yield very significant gains in power consumption.

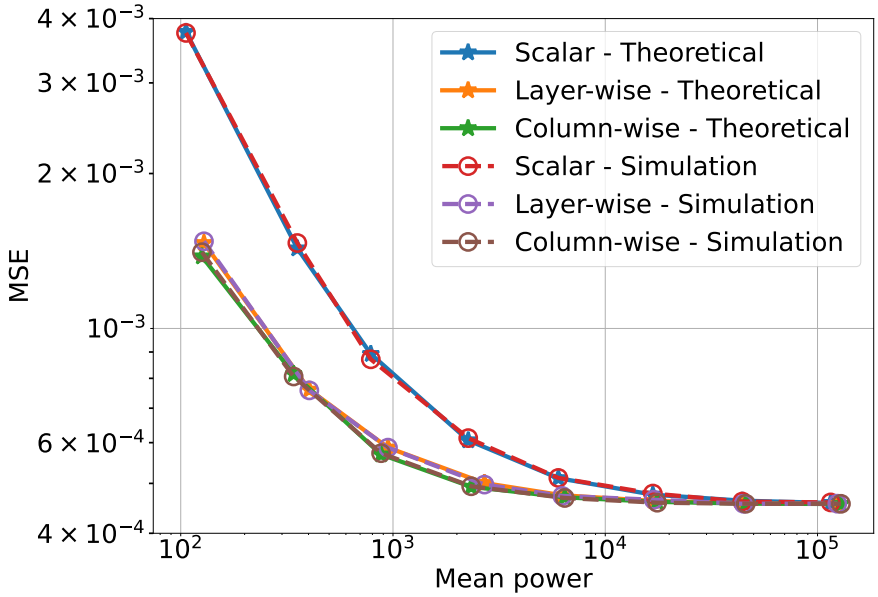



Figure 4.8 Minimized MSE for different cases of optimization and power constraints for the regression network

### Classification Problem

For our classification network, we first investigate if minimizing the MSE of the network is a good proxy for maximizing accuracy. To this end, we randomly generated different sets of $g_u$ in the columns-wise case and, for each set, we compute both the MSE and accuracy. Each $g_u$ set is represented by a point in Figure 4.9. We observe a general trend where as the maximum of the MSE decreases, the accuracy of the network increases. This suggests that reducing the MSE of the network is a good way to also improve its accuracy.

To illustrate how reducing the MSE can lead to improving the accuracy, we plot in Figure 4.10 the distribution of the outputs of the unreliable network for different inputs and values of $g_u$. Each histogram represents the output distribution for one of the classes and is shown in a different color, while the black lines show the outputs of a reliable network. From equation (4.7), we can see that the MSE can be decomposed as the sum of two terms: the variance of the erroneous output, and the squared difference between the mean of the incorrect output and the correct one.

In this Figure, we observe that when $g_u$ is low, the variances are high and the means of the network outputs move away from the correct values, which means that the MSE is high. This leads to a strong increase in the probability of a wrong classification, since there are many chances that the largest output value will belong to one of the incorrect classes due to the overlap between the outputs distributions. Conversely, when $g_u$ is high and the MSE

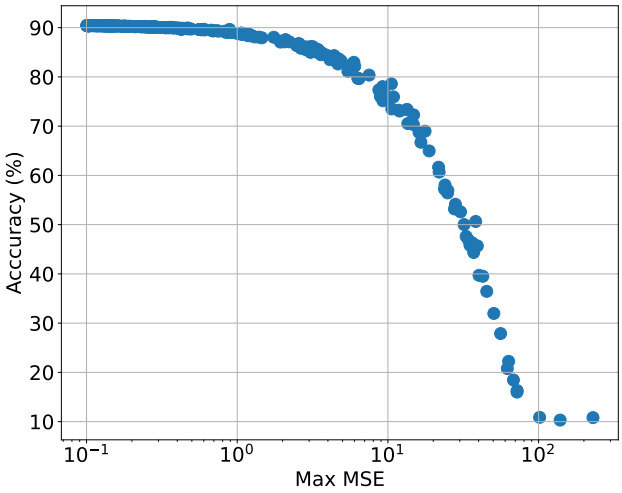



Figure 4.9 Accuracy depending on the maximum of the MSE for different sets of random $g_u$

is therefore low, the correct class outputs are completely separated from those of the other classes, with no overlap. As a result, there is no chance of a wrong classification. This shows how decreasing the MSE can lead to a better accuracy and how the parameter $g_u$ can lead to a reduced MSE by both decreasing the variance and decreasing the gap between the mean of the unreliable network and the correct output.

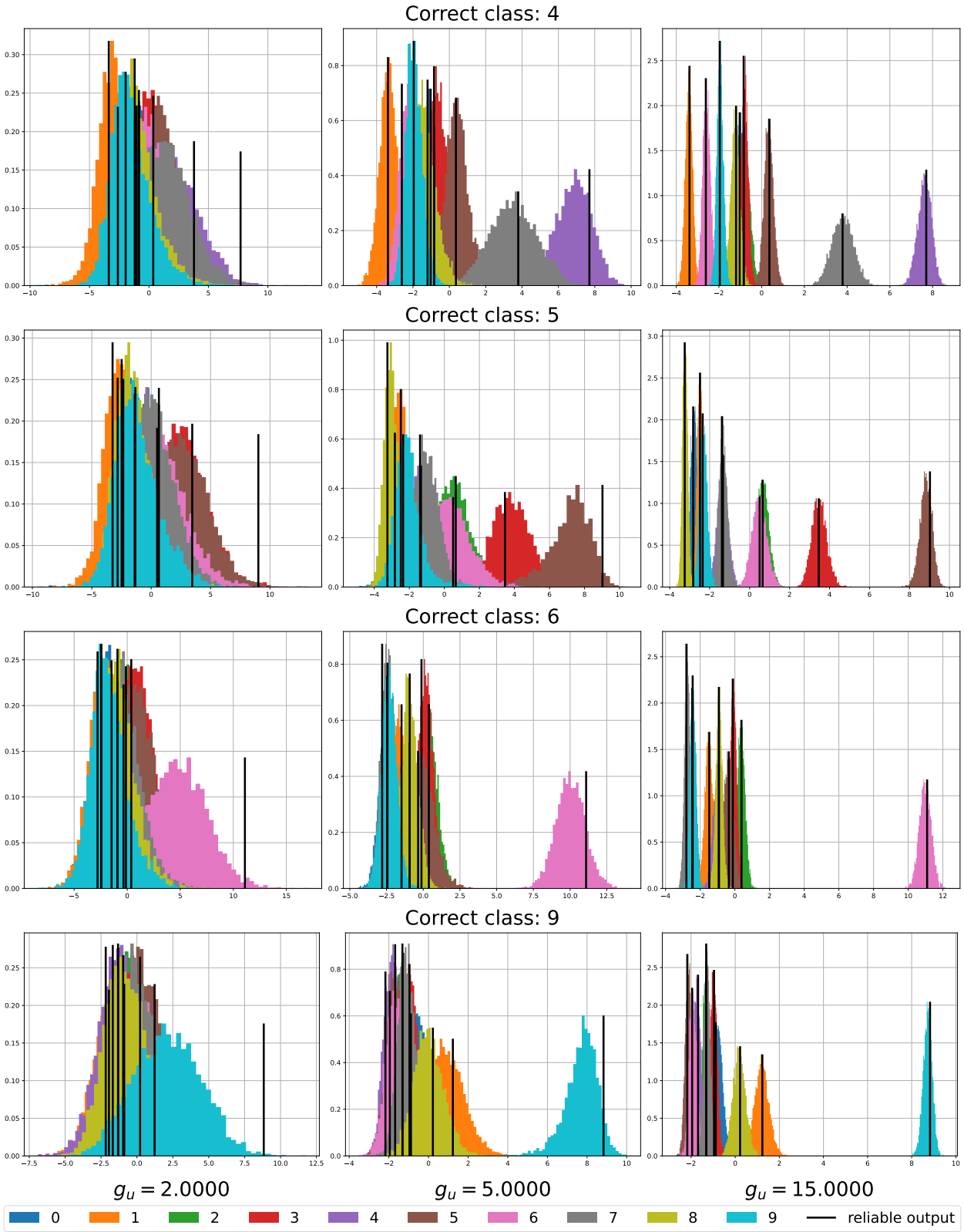



Figure 4.10 Distribution of the outputs of our unreliable CNN for different inputs and values of $g_u$

We now try to solve the optimization problem for our classification network. As in the regression problem, we select a subset of the CIFAR-10 data set of 100 samples (10 of each class) and use the genetic algorithm to find the best $g_u$ that minimizes the power usage for a given MSE constraint. Once the algorithm has converged to a solution for $g_u$, we use it to compute the accuracy of the network and its mean power usage over the whole test set. The results using different MSE constraints for the scalar design and the layer-wise design are presented in Figure 4.11. Just like for the regression network, preliminary results showed no differences between the layer-wise design and column-wise design. Due to the high computational cost of optimizing for the column-wise design in the case of a larger network, the data points for all different MSE constraints were not evaluated. As a result, the column-wise design is not depicted in the figure. We observe that having more degrees of freedom for the optimization by using the layer-wise design for $g_u$ permits to achieve lower power usage for the same network performance. For instance, there is a 6% difference in power usage between the layer-wise and single $g_u$ case when achieving an accuracy of 90%.

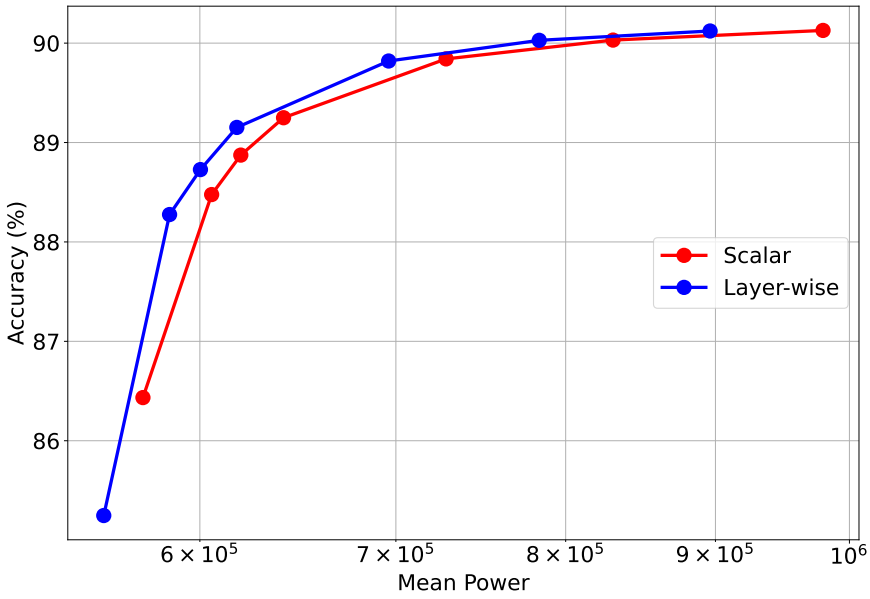



Figure 4.11 Accuracy after minimizing the MSE for different cases of optimization and MSE constraints for the classification network

## 4.7 Conclusion

In this chapter, we studied the mechanisms of error propagation in a memristor-based DNN using probabilistic analysis. This led us to a theoretical framework capable of accurately estimating the MSE of a high-performing, noisy network comprising a variety of layer types. We observed that such MSE estimation is significantly faster than a Monte-Carlo estimator. We further proposed two mappings for convolutional layers to memristor crossbars and compared their impact on the error. The accuracy of the method was validated on multiple tasks and networks and its speed was demonstrated for different confidence interval on the MSE estimate as well as well as several levels of noise. We have also shown how to use this approach to optimize the memristor hardware parameters with varying degrees of freedom, achieving significant power gains in both the regression and classification tasks.

The proposed framework benefits from a great versatility that can easily be adapted to other types of layers or error models. However, for some types of DNNs, this would require to enhance the analysis so that is capable of also estimating the MSE through a softmax activation function. This can be more challenging than for other activation functions such as ReLU due to the denominator of the softmax function which uses all inputs to compute each output. Taylor expansions for functions with multiple variables could be used but may lead to a large approximation error.

Albeit fast, the efficiency of our theoretical analysis implementation could still be improved by promoting better memory management and implementing dedicated software kernels, paving the way for applying it to larger networks and tasks.

# CHAPTER 5 CONCLUSION

## 5.1 Thesis Summary

In this thesis, we studied the implementation of two different algorithms on two different kinds of energy-efficient, unreliable memories: first, Kalman filtering using a voltage-scaled SRAM, and second, DNNs implemented on memristor crossbars. Despite the fact that we studied different algorithms for each system, we proposed a similar methodology in each case. We first specified our error model pertaining to the target unreliable hardware and how the algorithm is implemented on the unreliable hardware. For each model, we found that there is a set of parameters that can be used to balance the trade-off between the level of unreliability of the memory and its energy consumption. We then used these models to estimate the impact of the noise on the performance of the algorithm. For this, we provided analytical equations capable of estimating the MSE between the output of the algorithm implemented on unreliable hardware and a reliable implementation. These equations consist in propagating the estimation of the first and second order moments of the output of the successive computation steps of the studied algorithm. While Kalman filtering is composed only of linear operations, DNNs also use non-linear activation functions, which requires approximations for moment estimation. We then used these equations to estimate the performance degradation of the considered algorithm and its energy consumption. This allowed us to propose optimization methods to compute the best parameters capable of minimizing the energy consumption of the memory while achieving a certain target algorithm performance. In the case of Kalman filters, we were able to propose an analytical solution to the optimization problem while for DNNs, we provided a heuristic solution. In both instances, we performed extensive experiments to prove the accuracy of the proposed theoretical analysis against Monte-Carlo simulations. Furthermore, we were able to show that the solution to our optimization problems could lead to energy gains, with a gain of up to 50% for the Kalman filter case.

## 5.2 Perspectives

In this thesis, we aimed to introduce a generic methodology, which can then be adapted to other error models and algorithms. As a first straightfoward perspective, this methodology could be applied for instance to variations of the standard Kalman filter such as the extended Kalman Filter or the unscented Kalman filter which are capable of handling non-linearity

in the system models. Similarly, other types of DNNs could be studied under a memristor implementation, as our framework can easily be extended to other layer types.

The proposed method could also be improved in different ways. One of the main area for improvement would be to use more realistic error and energy models. In terms of error models, the analysis could be enriched by considering more error models such as cases where the noise is not independent and identically distributed or is spatially or temporally correlated. Concerning the energy usage, in Chapter 3 we only studied a simple model of the memory energy consumption but a more accurate model could take into account the exact number of read and write accesses to the memory. In addition, combining our theoretical method with a detailed circuit design could help to gain a better, more precise, understanding of the energy gains that can be achieved in practical implementations. Similarly, in Chapter 4 we looked only at the energy consumption of the actual memristor crossbars but we ignored the other components of the circuits such as digital-to-analog converters at the input of the crossbars and analog-to-digital converters at the output, or the digital systems doing the other operations of the network. This could be realized by the aid of existing energy simulators such as the one presented in [19] which is capable of estimating the energy consumption of a memristor-based DNN accelerator for a specified architecture.

Another possible improvement lies in the unreliable DNN considered in Chapter 4. The energy gain after optimization was lower than expected. This most probably means that computing the optimal scaling factor once the network is trained is not the best way to achieve significant energy gains. However, we may achieve better results by computing the scaling values directly during the training phase, along with the weights. Similarly, our MSE estimation framework could be integrated into the loss function of the network during training. As an alternative to noise injection, a term for minimizing the MSE computed from our theoretical analysis could be added to the loss function so that the network is trained to also minimize the error caused by the memristor noise. Nonetheless, implementing this requires to program custom kernels capable of backpropagating the gradients across our theoretical MSE estimation and for networks with large layer size, this could quickly lead to memory size issues.

An other way of using our framework on neural networks without requiring backpropagation rely on neural-architecture search (NAS). As mentioned in Chapter 2, to compensate for a degradation in performance we could aim to identify a network architecture which is inherently more robust to the system's unreliability. Therefore, we could associate the MSE estimation framework to a NAS method so that the MSE is computed for each possible network to find which one has the least error. This type of approach would be in line with

Hardware-Aware NAS, as our theoretical framework could be used to not only avoid performance degradation of a DNN but also minimize the energy footprint of the system for a chosen target hardware. Having a theoretical analysis could be used to find more efficiently the best DNN architecture capable of satisfying constraints related to the hardware model.

# REFERENCES


[1] J. Malmodin *et al.*, "Greenhouse gas emissions and operational electricity use in the ict and entertainment & media sectors," *J. of Ind. Ecol.*, vol. 14, no. 5, pp. 770–790, Oct. 2010.

[2] A. S. Andrae, "New perspectives on internet electricity use in 2030," *Eng. and Applied Science Lett.*, vol. 3, no. 2, pp. 19–31, Jun. 2020.

[3] J. Pullmann and D. Macko, "Increasing energy efficiency by minimizing collisions in long-range iot networks," in *2019 42nd Int. Conf. Telecommunications and Signal Process. (TSP)*, 2019, pp. 178–181.

[4] H. Jayakumar *et al.*, "Energy-efficient system design for iot devices," in *2016 21st Asia and South Pacific Des. Automat. Conf. (ASP-DAC)*, 2016, pp. 298–301.

[5] J. Yang *et al.*, "A low-power and portable biomedical device for respiratory monitoring with a stable power source," *Sensors*, vol. 15, no. 8, pp. 19 618–19 632, 2015.

[6] M. A. Hannan *et al.*, "Energy harvesting for the implantable biomedical devices: issues and challenges," *BioMedical Eng. OnLine*, vol. 13, no. 1, Jun. 2014.

[7] A. Espírito-Santo *et al.*, "The need for standardisation in low power smart sensing," in *IECON 2018 - 44th Annu. Conf. IEEE Ind. Electron. Soc.*, 2018, pp. 3870–3875.

[8] A. Fayyazi *et al.*, "An ultra low-power memristive neuromorphic circuit for internet of things smart sensors," *IEEE Internet of Things J.*, vol. 5, no. 2, pp. 1011–1022, 2018.

[9] X. Xu *et al.*, "Scaling for edge inference of deep neural networks," *Nature Electron.*, vol. 1, no. 4, pp. 216–222, Apr. 2018.

[10] A. Pedram *et al.*, "Dark Memory and Accelerator-Rich System Optimization in the Dark Silicon Era," *IEEE Des. Test*, vol. 34, no. 2, pp. 39–50, Apr. 2017.

[11] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. Dig. of Tech. Papers (ISSCC)*, Feb. 2014, pp. 10–14.

[12] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *J. of Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[13] A. D. Mauro *et al.*, "Always-on 674$\mu$W@4GOP/s error resilient binary neural networks with aggressive SRAM voltage scaling on a 22-nm IoT end-node," *IEEE Trans. Circuits and Syst. I: Regular Papers*, vol. 67, no. 11, pp. 3905–3918, 2020.

[14] W. J. Dally, R. C. Harting, and T. M. Aamodt, *Digital Design Using VHDL: A Systems Approach*. Cambridge, United Kingdom: Cambridge University Press, 2015.

[15] Y. Kim *et al.*, "Generalized Water-Filling for Source-Aware Energy-Efficient SRAMs," *IEEE Trans. Commun.*, vol. 66, no. 10, pp. 4826–4841, 2018.

[16] L. O. Chua, "Memristor – The Missing Circuit Element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[17] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, Jul. 2020.

[18] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Technical report, University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009, backup Publisher: University of Toronto.

[19] Y. N. Wu, V. Sze, and J. S. Emer, "An architecture-level energy and area estimator for processing-in-memory accelerator designs," in *2020 IEEE Int. Symp. Performance Analysis of Syst. and Software (ISPASS)*, 2020, pp. 116–118.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Adv. in Neural Inf. Process. Syst.*, F. Pereira *et al.*, Eds., vol. 25. Curran Associates, Inc., 2012.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd Int. Conf. Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[22] C. Szegedy *et al.*, "Inception-v4, inception-ResNet and the impact of residual connections on learning," *AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017.

[23] C. Liu *et al.*, "Progressive neural architecture search," *CoRR*, vol. abs/1712.00559, 2017. [Online]. Available: http://arxiv.org/abs/1712.00559

[24] H. Touvron *et al.*, "Fixing the train-test resolution discrepancy," in *Adv. in Neural Inf. Process. Syst.*, H. Wallach *et al.*, Eds., vol. 32. Curran Associates, Inc., 2019.

[25] Q. Xie *et al.*, "Self-training with noisy student improves ImageNet classification," in *2020 IEEE/CVF Conf. on Comput. Vision and Pattern Recognit. (CVPR)*. IEEE, Jun. 2020.

[26] X. Zhai *et al.*, "Scaling vision transformers," in *2022 IEEE/CVF Conf. Comput. Vision and Pattern Recognit. (CVPR)*. IEEE, Jun. 2022.

[27] J. Yu *et al.*, "Coca: Contrastive captioners are image-text foundation models," *Transactions on Machine Learning Research*, 2022.

[28] R. G. Dreslinski *et al.*, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proc. IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.

[29] K. V. Pham *et al.*, "Asymmetrical Training Scheme of Binary-Memristor-Crossbar-Based Neural Networks for Energy-Efficient Edge-Computing Nanoscale Systems," *Micromachines (Basel)*, vol. 10, no. 2, Feb. 2019.

[30] K. Sung and H. Kim, "Simplified KF-based energy-efficient vehicle positioning for smartphones," *J. of Commun. and Networks*, vol. 22, no. 2, pp. 93–107, Apr. 2020.

[31] G. Anania *et al.*, "Development of a novel algorithm for human fall detection using wearable sensors," in *IEEE SENSORS*, Oct. 2008, pp. 1336–1339.

[32] H. Kaul *et al.*, "Near-threshold voltage (NTV) design — Opportunities and challenges," in *DAC Des. Automat. Conf. 2012*, Jun. 2012, pp. 1149–1154.

[33] D. Markovic *et al.*, "Ultralow-Power Design in Near-Threshold Region," *Proc. IEEE*, vol. 98, no. 2, pp. 237–252, Feb. 2010.

[34] A. Chatterjee and L. R. Varshney, "Energy-reliability limits in nanoscale circuits," in *2016 Inf. Theory and Applications Workshop (ITA)*, Jan. 2016, pp. 1–6.

[35] M. A. Zidan *et al.*, "Memristor-based memory: The sneak paths problem and solutions," *Microelectron. J.*, vol. 44, no. 2, pp. 176–183, 2013.

[36] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *2015 , Automat. Test in Europe Conf. Exhibition (DATE)*, Mar. 2015, pp. 1718–1725.

[37] S. Gi *et al.*, "Fundamental issues of implementing hardware neural networks using memristor," in *2015 Int. SoC Des. Conf. (ISOCC)*, Nov. 2015, pp. 215–216.

[38] Y. Li *et al.*, "Review of memristor devices in neuromorphic computing: materials sciences and device challenges," *J. of Physics D: Applied Physics*, vol. 51, no. 50, p. 503002, Sep. 2018.

[39] C. Baeumer *et al.*, "Subfilamentary networks cause cycle-to-cycle variability in memristive devices," *ACS Nano*, vol. 11, no. 7, pp. 6921–6929, Jul. 2017.

[40] Z. Chen, C. Schoeny, and L. Dolecek, "Hamming Distance Computation in Unreliable Resistive Memory," *IEEE Trans. Commun.*, vol. 66, no. 11, pp. 5013–5027, Nov. 2018.

[41] H. Lv *et al.*, "Evolution of conductive filament and its impact on reliability issues in oxide-electrolyte based resistive random access memory," *Scientific Reports*, vol. 5, no. 1, Jan. 2015.

[42] X. Zhao *et al.*, "Breaking the current-retention dilemma in cation-based resistive switching devices utilizing graphene with controlled defects," *Advanced Materials*, vol. 30, no. 14, p. 1705193, 2018.

[43] H. Li *et al.*, "Memristive crossbar arrays for storage and computing applications," *Advanced Intelligent Syst.*, vol. 3, no. 9, 2021.

[44] S. Liu *et al.*, "A Memristor-Based Optimization Framework for Artificial Intelligence Applications," *IEEE Circuits and Syst. Magazine*, vol. 18, no. 1, pp. 29–44, 2018.

[45] Y. Zhang, X. Wang, and E. G. Friedman, "Memristor-Based Circuit Design for Multilayer Neural Networks," *IEEE Trans. Circuits and Syst. I: Regular Papers*, vol. 65, no. 2, pp. 677–686, Feb. 2018.

[46] M. Hu *et al.*, "Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[47] R. Hasan, T. M. Taha, and C. Yakopcic, "On-chip training of memristor crossbar based multi-layer neural networks," *Microelectron.J.*, vol. 66, pp. 31–40, Aug. 2017.

[48] P. Yao *et al.*, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, Jan. 2020.

[49] P.-F. Chiu *et al.*, "A Binarized Neural Network Accelerator with Differential Crosspoint Memristor Array for Energy-Efficient MAC Operations," in *2019 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, May 2019, pp. 1–5.

[50] M. Cui and Y. Zhang, "Memristive Synaptic Circuits for Deep Convolutional Neural Networks," in *2019 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, May 2019, pp. 1–5.

[51] Y. Zhang *et al.*, "Memristive Quantized Neural Networks: A Novel Approach to Accelerate Deep Learning On-Chip," *IEEE Trans. Cybern.*, pp. 1–13, 2019.

[52] T. Hirtzlin *et al.*, "Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks," in *2019 IEEE Int. Conf. Artificial Intell. Circuits and Syst. (AICAS)*, Mar. 2019, pp. 288–292.

[53] O. Krestinskaya, K. N. Salama, and A. P. James, "Learning in Memristive Neural Network Architectures Using Analog Backpropagation Circuits," *IEEE Trans. Circuits and Syst. I: Regular Papers*, vol. 66, no. 2, pp. 719–732, Feb. 2019.

[54] K. Smagulova, O. Krestinskaya, and A. P. James, "A memristor-based long short term memory circuit," *Analog Integr Circ Sig Process.*, vol. 95, no. 3, pp. 467–472, Jun. 2018.

[55] M. S. Alam *et al.*, "Memristor Based Autoencoder for Unsupervised Real-Time Network Intrusion and Anomaly Detection," in *Proc. Int. Conf. Neuromorphic Syst.*, ser. ICONS '19. Association for Computing Machinery, Jul. 2019, pp. 1–8.

[56] S. Jain *et al.*, "RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars," *arXiv:1809.00072 [cs]*, Jun. 2020, arXiv: 1809.00072. [Online]. Available: http://arxiv.org/abs/1809.00072

[57] M. J. Rasch *et al.*, "A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays," in *2021 IEEE 3rd Int. Conf. Artif. Intell. Circuits and Syst. (AICAS)*, 2021, pp. 1–4.

[58] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conf. Comput. Vision and Pattern Recognit.* Miami, FL: IEEE, Jun. 2009, pp. 248–255.

[59] O. B. Usman *et al.*, "Mmse precoder for massive mimo using 1-bit quantization," in *2016 IEEE Int. Conf. Acoustics, Speech and Signal Process. (ICASSP)*, 2016, pp. 3381–3385.

[60] Y. Pan, X. Mao, and P. Qiu, "Energy-efficient quantization and transmission in distributed estimation," in *2010 5th Int. ICST Conf. Commun. and Networking in China*, 2010, pp. 1–5.

[61] X. Luo and G. Giannakis, "Energy-constrained optimal quantization for wireless sensor networks," in *2004 1st Annu. IEEE Commun. Soc. Conf. Sensor and Ad Hoc Commun. and Networks*, 2004, pp. 272–278.

[62] J.-H. Kwon and E.-J. Kim, "Quantization-based data size reduction for energy-efficient internet of medical things," in *2022 Int. Conf. Power, Energy and Innovations (ICPEI)*, 2022, pp. 1–4.

[63] J. Park, J. H. Choi, and K. Roy, "Dynamic Bit-Width Adaptation in DCT: An Approach to Trade Off Image Quality and Computation Energy," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 18, no. 5, pp. 787–793, May 2010.

[64] M. Courbariaux *et al.*, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv:1602.02830 [cs]*, Mar. 2016, arXiv: 1602.02830.

[65] I. Hubara *et al.*, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *arXiv:1609.07061 [cs]*, Sep. 2016, arXiv: 1609.07061.

[66] B. Moons *et al.*, "Minimum energy quantized neural networks," *2017 51st Asilomar Conf. Signals, Syst., and Computers*, 2017.

[67] X. Sun *et al.*, "Ultra-low precision 4-bit training of deep neural networks," in *Advances in Neural Information Processing Systems*, H. Larochelle *et al.*, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1796–1807.

[68] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symp. (ETS)*, May 2013, pp. 1–6, iSSN: 1558-1780.

[69] A. Balatsoukas-Stimming and A. Burg, "Density evolution for min-sum decoding of ldpc codes under unreliable message storage," *IEEE Commun. Lett.*, vol. 18, no. 5, pp. 849–852, 2014.

[70] K. Le *et al.*, "Efficient hardware implementation of probabilistic gradient descent bit-flipping," *IEEE Trans. Circuits and Syst. I: Regular Papers*, vol. 64, no. 4, pp. 906–917, 2017.

[71] J. Geldmacher and J. Götze, "On fault tolerant decoding of turbo codes," in *2012 7th Int. Symp. Turbo Codes and Iterative Inf. Process. (ISTC)*, 2012, pp. 245–249.

[72] S. H. Nawab *et al.*, "Approximate signal processing," *The J. of VLSI Signal Process.*, vol. 15, no. 1/2, pp. 177–200, 1997.

[73] V. K. Chippa *et al.*, "Analysis and characterization of inherent application resilience for approximate computing," in *2013 50th ACM/EDAC/IEEE Des. Automat. Conf. (DAC)*, May 2013, pp. 1–9.

[74] C. Chekuri and S. Gupta, "Perturbation resilient clustering for k-center and related problems via lp relaxations," in *Int. Workshop and Int. Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techn.*, 2018.

[75] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise," in *Asian Conf. Machine Learning*, 2011.

[76] L. Yang and B. Murmann, "Sram voltage scaling for energy-efficient convolutional neural networks," in *18th Int. Symp. Quality Electronic Des. (ISQED)*, Mar. 2017, pp. 7–12.

[77] J.-C. Vialatte and F. Leduc-Primeau, "A Study of Deep Learning Robustness Against Computation Failures," *arXiv:1704.05396 [cs]*, Apr. 2017, arXiv: 1704.05396.

[78] G. B. Hacene *et al.*, "Training Modern Deep Neural Networks for Memory-Fault Robustness," in *IEEE Int. Symp. Circuits and Syst. (ISCAS)*, May 2019, pp. 1–5.

[79] L. Yang *et al.*, "Bit error tolerance of a cifar-10 binarized convolutional neural network processor," in *2018 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, 2018, pp. 1–5.

[80] H. Chen, P. K. Varshney, and J. H. Michels, "Noise Enhanced Parameter Estimation," *IEEE Trans. Signal Process.*, vol. 56, no. 10, pp. 5074–5081, Oct. 2008.

[81] H. Chen, L. R. Varshney, and P. K. Varshney, "Noise-Enhanced Information Systems," *Proc. IEEE*, vol. 102, no. 10, pp. 1607–1621, Oct. 2014.

[82] B. Franzke and B. Kosko, "Noise can speed convergence in Markov chains," *Physical Review E*, vol. 84, no. 4, p. 041112, Oct. 2011.

[83] K. Audhkhasi, O. Osoba, and B. Kosko, "Noise-enhanced convolutional neural networks," *Neural Networks*, vol. 78, pp. 15–23, Jun. 2016.

[84] R. Hegde and N. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proceedings. 1999 Int. Symp. Low Power Electron. and Des. (Cat. No.99TH8477)*, Aug. 1999, pp. 30–35.

[85] R. Hegde and N. Shanbhag, "Soft digital signal processing," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 9, no. 6, pp. 813–823, 2001.

[86] R. A. Abdallah and N. R. Shanbhag, "Error-resilient low-power viterbi decoders," in *Proc. 13th Int. Symp. Low power Electron. and Des. (ISLPED '08)*, 2008, pp. 111–116.

[87] B. Shim, S. Sridhara, and N. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 12, no. 5, pp. 497–510, May 2004.

[88] S. Zhang and N. R. Shanbhag, "Embedded error compensation for energy efficient DSP systems," in *2014 IEEE Global Conf. Signal and Inf. Process. (GlobalSIP)*, Dec. 2014, pp. 30–34.

[89] C. Hadjicostis, "Nonconcurrent error detection and correction in fault-tolerant linear finite-state machines," *IEEE Trans. Automatic Control*, vol. 48, no. 12, pp. 2133–2140, 2003.

[90] C. Hadjicostis and G. Verghese, "Coding approaches to fault tolerance in linear dynamic systems," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 210–228, 2005.

[91] Y. Yang, P. Grover, and S. Kar, "Computing Linear Transformations With Unreliable Components," *IEEE Trans. Inf. Theory*, vol. 63, no. 6, pp. 3729–3756, Jun. 2017.

[92] E. Dupraz and L. R. Varshney, "Binary Recursive Estimation on Noisy Hardware," in *2019 IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 877–881.

[93] Y. Lin, S. Zhang, and N. R. Shanbhag, "Variation-Tolerant Architectures for Convolutional Neural Networks in the Near Threshold Voltage Regime," in *2016 IEEE Int. Workshop on Signal Process. Syst. (SiPS)*, Oct. 2016, pp. 17–22.

[94] Y. Lin and J. R. Cavallaro, "Energy-efficient Convolutional Neural Networks via Statistical Error Compensated Near Threshold Computing," in *2018 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, May 2018, pp. 1–5, iSSN: 2379-447X.

[95] S. Kim *et al.*, "Energy-Efficient Neural Network Acceleration in the Presence of Bit-Level Memory Errors," *IEEE Trans. Circuits and Syst. I: Regular Papers*, vol. 65, no. 12, pp. 4285–4298, Dec. 2018.

[96] V. Joshi *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nature Commun.*, vol. 11, no. 1, p. 2473, Dec. 2020.

[97] M. J. Rasch *et al.*, "Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators," 2023.

[98] S. Henwood, F. Leduc-Primeau, and Y. Savaria, "Layerwise Noise Maximisation to Train Low-Energy Deep Neural Networks," in *2nd IEEE Int. Conf. Artificial Intell. Circuits and Syst. (AICAS)*, Aug. 2020, pp. 271–275.

[99] P. Dey *et al.*, "Regularizing multilayer perceptron for robustness," *IEEE Trans. Syst., Man, and Cybern.: Syst.*, vol. 48, no. 8, pp. 1255–1266, Aug. 2018.

[100] S. Buschjäger *et al.*, "Margin-maximization in binarized neural networks for optimizing bit error tolerance," in *2021 Des., Automat. & Test in Europe Conf. & Exhibition (DATE)*, 2021, pp. 673–678.

[101] G. Mordido, S. Chandar, and F. Leduc-Primeau, "Sharpness-aware training for accurate inference on noisy DNN accelerators," *arXiv preprint arXiv:2211.11561*, 2022.

[102] N. S. Keskar *et al.*, "On large-batch training for deep learning: Generalization gap and sharp minima," in *Int. Conf. Learning Representations*, 2017.

[103] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[104] H. Benmeziane *et al.*, "Hardware-aware neural architecture search: Survey and taxonomy," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed. International Joint Conferences on Artificial Intelligence Organization, aug 2021, pp. 4322–4329, survey Track.

[105] K. T. Chitty-Venkata and A. K. Somani, "Neural architecture search survey: A hardware perspective," *ACM Comput. Surv.*, vol. 55, no. 4, Nov. 2022.

[106] E. Dupraz, L. R. Varshney, and F. Leduc-Primeau, "Power-Efficient Deep Neural Networks with Noisy Memristor Implementation," in *2021 IEEE Inf. Theory Workshop (ITW)*, Oct. 2021, pp. 1–5.

[107] X. Lai *et al.*, "IoT Implementation of Kalman Filter to Improve Accuracy of Air Quality Monitoring and Prediction," *Applied Sciences*, vol. 9, no. 9, p. 1831, Jan. 2019.

[108] T. Wang *et al.*, "Forest fire detection system based on Fuzzy Kalman filter," in *2020 Int. Conf. Urban Eng. and Management Science (ICUEMS)*, Apr. 2020, pp. 630–633.

[109] E. E. Yaz, C. S. Jeong, and Y. I. Yaz, "An LMI approach to discrete-time observer design with stochastic resilience," *J. of Computational and Applied Mathematics*, vol. 188, no. 2, pp. 246–255, Apr. 2006.

[110] Y. Chen, C. Chen, and A. Xue, "Distributed non-fragile $l_2 - l_\infty$ filtering over sensor networks with random gain variations and fading measurements," *Neurocomputing*, vol. 338, pp. 154–162, Apr. 2019.

[111] N. Nahi, "Optimal recursive estimation with uncertain observation," *IEEE Trans. Inf. Theory*, vol. 15, no. 4, pp. 457–462, Jul. 1969.

[112] F. O. Hounkpevi and E. E. Yaz, "Robust minimum variance linear state estimators for multiple sensors with different failure rates," *Automatica*, vol. 43, no. 7, pp. 1274–1280, Jul. 2007.

[113] I. R. Petersen, D. C. McFarlane, and M. A. Rotea, "Optimal Guaranteed Cost Control of Discrete-time Uncertain Linear Systems," *IFAC Proceedings Volumes*, vol. 26, no. 2, Part 2, pp. 35–38, Jul. 1993.

[114] G.-H. Yang and J. L. Wang, "Robust nonfragile Kalman filtering for uncertain linear systems with estimator gain uncertainty," *IEEE Trans. Automatic Control*, vol. 46, no. 2, pp. 343–348, Feb. 2001.

[115] Y. Huang *et al.*, "A Novel Adaptive Kalman Filter With Inaccurate Process and Measurement Noise Covariance Matrices," *IEEE Trans. Automatic Control*, vol. 63, no. 2, pp. 594–601, Feb. 2018.

[116] A. Jarrah, "Optimized parallel architecture of Kalman filter for radar tracking applications," *Jordan J. of Elect. Eng.*, vol. 2, no. 3, pp. 215–230, May 2016.

[117] T. Sunil Kumar and P. Duraiswamy, "Optimization of Kalman Filter for Target Tracking Applications," in *Advances in Multidisciplinary Analysis and Optimization*, ser. Lecture Notes in Mechanical Engineering, R. R. Salagame *et al.*, Eds. Singapore: Springer, 2020, pp. 203–212.

[118] P. T. L. Pereira *et al.*, "Exploring Architectural Solutions for an Energy-Efficient Kalman Filter Gain Realization," in *26th IEEE Int. Conf. Electron., Circuits and Syst. (ICECS)*, Nov. 2019, pp. 650–653.

[119] A. B. Stripad, "Performance Degradation in Digitally Implemented Kalman Filters," *IEEE Trans. Aerosp. and Electron. Syst.*, vol. AES-17, no. 5, pp. 626–634, Sep. 1981.

[120] M. Verhaegen and P. V. Dooren, "Numerical aspects of different Kalman filter implementations," *IEEE Trans. Automatic Control*, vol. 31, no. 10, pp. 907–917, 1986.

[121] S. Sun *et al.*, "Quantized Kalman Filtering," in *IEEE 22nd Int. Symp. Intelligent Control*, Oct. 2007, pp. 7–12.

[122] D. Li, S. Kar, and S. Cui, "Distributed Kalman Filtering with quantized sensing state," in *IEEE Int. Conf. Acoustics, Speech and Signal Process. (ICASSP)*, Apr. 2015, pp. 4040–4044.

[123] X. Hu *et al.*, "Quantized Kalman Filter Tracking in Directional Sensor Networks," *IEEE Trans. Mobile Computing*, vol. 17, no. 4, pp. 871–883, Apr. 2018.

[124] K. You *et al.*, "Quantized filtering of linear stochastic systems," *Transactions of the Institute of Measurement and Control*, vol. 33, no. 6, pp. 683–698, Aug. 2011.

[125] K. You, Y. Zhao, and L. Xie, "Recursive quantized state estimation of discrete-time linear stochastic systems," in *7th Asian Control Conf.*, Aug. 2009, pp. 170–175.

[126] Y. Bar-Shalom, X.-R. Li, and T. Kirubarajan, *State Estimation in Discrete-Time Linear Dynamic Systems*. John Wiley & Sons, Ltd, 2002, ch. 5, pp. 199–266.

[127] N. Thacker and A. Lacey, "Tutorial: The kalman filter," *Imaging Science and Biomedical Eng. Division, Medical School, University of Manchester*, p. 61, 1998.

[128] J. Ziv, "On universal quantization," *IEEE Trans. Inf. Theory*, vol. 31, no. 3, pp. 344–347, May 1985.

[129] A. Sripad and D. Snyder, "A necessary and sufficient condition for quantization errors to be uniform and white," *IEEE Trans. Acoustics, Speech, and Signal Process.*, vol. 25, no. 5, pp. 442–448, Oct. 1977.

[130] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS," *IEEE Trans. Computer-Aided Des. of Integr. Circuits and Syst.*, vol. 24, no. 12, pp. 1859–1880, Dec. 2005.

[131] E. Dupraz *et al.*, "Analysis and Design of Finite Alphabet Iterative Decoders Robust to Faulty Hardware," *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2797–2809, Aug. 2015.

[132] C. Kameni Ngassa *et al.*, "Density Evolution and Functional Threshold for the Noisy Min-Sum Decoder," *IEEE Trans. Commun.*, vol. 63, no. 5, pp. 1497–1509, May 2015.

[133] D. Berberidis and G. B. Giannakis, "Data Sketching for Large-Scale Kalman Filtering," *IEEE Trans. Signal Process.*, vol. 65, no. 14, pp. 3688–3701, Jul. 2017.

[134] G. Pedretti and D. Ielmini, "In-Memory Computing with Resistive Memory Circuits: Status and Outlook," *Electron.*, vol. 10, no. 9, p. 1063, Jan. 2021.

[135] J. Wen *et al.*, "Behavioral Model of Dot-Product Engine Implemented with 1T1R Memristor Crossbar Including Assessment," in *2021 24th Int. Symp. Des. and Diagnostics of Electron. Circuits Syst. (DDECS)*, Apr. 2021, pp. 29–32.

[136] A. J. Pérez-Ávila *et al.*, "Behavioral modeling of multilevel HfO2-based memristors for neuromorphic circuit simulation," in *2020 XXXV Conf. Des. of Circuits and Integr. Syst. (DCIS)*, Nov. 2020, pp. 1–6.

[137] S. Moon, K. Shin, and D. Jeon, "Enhancing Reliability of Analog Neural Network Processors," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 27, no. 6, pp. 1455–1459, Jun. 2019.

[138] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *2016 ACM/IEEE 43rd Annu. Int. Symp. Computer Architecture (ISCA)*. Seoul, South Korea: IEEE, Jun. 2016, pp. 14–26.

[139] C. Li *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature Commun.*, vol. 9, no. 1, p. 2385, Jun. 2018.

[140] C. Li *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electron.*, vol. 1, no. 1, pp. 52–59, Jan. 2018.

[141] D. Dera *et al.*, "PremiUm-CNN: Propagating uncertainty towards robust convolutional neural networks," *IEEE Trans. Signal Process.*, vol. 69, pp. 4669–4684, 2021.

[142] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *2016 53nd ACM/EDAC/IEEE Des. Automat. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[143] V. Milo *et al.*, "Multilevel HfO2-based RRAM devices for low-power neuromorphic networks," *APL Materials*, vol. 7, no. 8, p. 081120, Aug. 2019.

[144] Z. Wang *et al.*, "In situ training of feed-forward and recurrent convolutional memristor networks," *Nature Machine Intell.*, vol. 1, no. 9, pp. 434–442, Sep. 2019.

[145] T. Gokmen, M. Onen, and W. Haensch, "Training Deep Convolutional Neural Networks with Resistive Cross-Point Devices," *Frontiers in Neuroscience*, vol. 11, p. 538, Oct. 2017.

[146] D. B. Owen, "A table of normal integrals," *Commun. in Statist. - Simul. and Comput.*, vol. 9, no. 4, pp. 389–419, 1980.

[147] T. Chen *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Conf. Operating Syst. Des. and Implementation*, 2018.

[148] W. Niu *et al.*, "DNNFusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proc. 42nd ACM SIGPLAN Int. Conf. Program. Lang. Des. and Implementation*, Jun. 2021.

[149] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, 1st ed., ser. Complex adaptive systems. Cambridge, Mass: MIT Press, 1992.

[150] A. Coraddu *et al.*, "Machine learning approaches for improving condition-based maintenance of naval propulsion plants," *Proc. Institution of Mechanical Eng., Part M: J. of Eng. for the Maritime Environ.*, vol. 230, no. 1, pp. 136–153, Feb. 2016.

[151] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[152] M. R. Driels and Y. S. Shin, "Determining the number of iterations for Monte Carlo simulations of weapon effectiveness," Monterey, California. Naval Postgraduate School, Tech. Rep., 2004.