

Population coding to improve fault tolerance of neuromorphic networks in regression tasks

Alexis Gleyo

Université de Lorraine, CNRS, LORIA
F-54000 Nancy, France

Bernard Girau

Université de Lorraine, CNRS, LORIA
F-54000 Nancy, France

Abstract—Spiking Neural Networks (SNNs) and specialized neuromorphic hardware represent a promising prospect for energy-efficient computation. However, this hardware is susceptible to permanent faults, such as dead or saturated neurons, which can compromise the model’s reliability. As semiconductor technologies advance toward ever-smaller feature sizes, process variations and defect rates increase, making fault tolerance a critical requirement. The intrinsic robustness of neural computation, inspired by biological systems, offers an opportunity to develop more sustainable neuromorphic design practices—by enabling the use of partially defective chips both at manufacturing time and during long-term deployment. In this context, we argue that population coding provides an additional layer of fault resilience, as it allows neural models to tolerate hardware-level defects without requiring retraining or architectural modifications. This paper investigates the inherent and passive fault tolerance conferred by population coding as a robustness strategy in regression tasks. We propose a methodology where continuous variables are represented using Gaussian Receptive Field (GRF) population encoding and decoded from the SNN’s output using a Maximum Likelihood Estimation (MLE) method designed to mitigate the influence of faulty neurons.

We systematically evaluate this approach through fault injection experiments by introducing an increasing number of faults across different network layers. Our results demonstrate that population-coded models may be significantly more resilient to permanent faults than those using a direct single-neuron output. This work validates that population coding provides a powerful architecture for fault-tolerant neuromorphic systems without the overhead of active fault detection and reconfiguration hardware.

Index Terms—fault tolerance, spiking neural networks, population coding

I. INTRODUCTION

Spiking Neural Networks (SNNs) mimic the communication of biological neurons using discrete signals called spikes. SNNs appear as a promising low-power alternative to conventional artificial neural networks [11]. Their energy-efficiency makes them convenient for deployment in resource-constrained settings and on specialized neuromorphic hardware designed for fast, low-power computation. SNNs are often portrayed as being the upcoming “third generation” of neural networks [13].

However, the specialized digital hardware essential for running SNNs is susceptible to permanent faults, whether from manufacturing defects or environmental factors like radiation and electromagnetic interference [19]. Nevertheless, beyond energy efficiency and task performance, neural computation is often assumed to inherit several desirable properties from its biological inspiration — among them, a certain tolerance to imprecision, noise, and hardware faults [2]. The brain relies

on massively parallel and distributed processing among slow, unreliable, and noisy components, and yet it exhibits a high level of resilience and adaptability [12], [21]. This is why it is often expected that artificial neural networks might exhibit similar intrinsic fault tolerance, due to their inherent redundancy and distributed structure. Being even more biologically inspired than second generation neural networks, SNNs should have an even higher level of fault tolerance.

However, this assumption does not always hold in practice. Studies have shown that many neural networks are not inherently fault tolerant without appropriate architectural or algorithmic choices (e.g. [14] for deep NNs, [20], [22] for SNNs). Simply mimicking biological systems is not sufficient to ensure robustness against physical degradation. This issue is becoming increasingly challenging as advances in semiconductor manufacturing push towards nanoscale integration, where process variations, thermal effects, and aging will decrease the probability of producing and using perfectly functional circuits. In this context, designing models and architectures that remain functional on partially defective hardware may become a necessity. Thus, the ability to develop neural algorithms that can exploit the computing potential of unreliable substrates appears to be a major argument in favor of neuromorphic systems, since it may help reduce the amount of discarded silicon, both at manufacturing and during long-term deployment, and thus contribute to more sustainable design and production practices.

In this paper, we focus on a specific property observed in biological neural populations: population coding. We argue that population coding not only supports efficient and robust information representation [6], but also contributes to fault tolerance by distributing information across many units. As such, it allows SNNs to maintain functional performance even in the presence of neuron-level defects, without requiring retraining or explicit error correction.

In this work, we investigate how population coding can be leveraged to enhance the inherent robustness of multilayered SNNs in regression tasks, particularly under permanent neuron faults in potentially imperfect neuromorphic hardware (simulated in software). We implement such population coding via Gaussian Receptive Fields (GRFs) combined with a Maximum Likelihood Estimation (MLE) decoding method. By comparing population-based and direct-output models through systematic fault injection experiments, we aim to assess our claim that population coding offers a low-overhead path toward more fault-

resilient neuromorphic computation. The paper is organized as follows: section II reviews related work on fault tolerance in neural networks and SNNs; section III details the methodology of our study; section IV presents our experimental setup and results, before we discuss them in section V.

II. RELATED WORK

Population coding offers a more resilient way to represent information. By distributing information over several neurons, the network becomes more robust to the failure of individual neurons. This principle has been demonstrated in Artificial Neural Networks (ANNs), where population-coded networks have better robustness against input noise than their direct network counterparts [8].

In the domain of Spiking Neural Networks, this robustness is directly inspired by the brain’s own resilience to neuronal loss and damage. Our work aims to provide a systematic analysis and direct demonstration of how population coding mitigates the impact of physical hardware failures in SNNs.

Although the fault tolerance of SNN architectures has been studied in works such as [22], [17], to our knowledge, it has yet to be studied in the context of regression tasks, for which population coding is a relevant coding choice, as illustrated by [23], where the authors propose a SNN version of Regression As Classification (RAC) [7], which is similar to population coding because of the distributed nature of its output.

In [17], the authors propose a methodology to mitigate the impact of faulty neurons. This method focuses on identifying faulty neurons and the severity of their fault in order to prune or adapt the network efficiently. While effective, this method requires additional overhead for fault detection and network reconfiguration. In contrast, our work explores a passive fault tolerance strategy. Rather than detecting and reconfiguring the network, we use the properties of Gaussian Receptive Field (GRF) population coding combined with a Maximum Likelihood Estimation (MLE) decoder. This approach inherently tolerates faults by making a statistically sound inference from the entire, potentially corrupted, population output.

III. METHODOLOGY

We chose to restrict our study to very common and simple technological choices for SNN architectures, in order to minimize the bias such choices may have on the results. We thus use multilayer spiking neural networks (spiking versions of classical multilayer perceptrons) with standard leaky integrate-and-fire neurons (a very common spiking neuron model, often implemented as the basic neural block in neuromorphic chips). Mechanisms of population encoding are then added to this simple architecture.

A. The Leaky Integrate-and-Fire (LIF) model

The Leaky Integrate-and-Fire (LIF) model is one of the most common models of spiking neurons used in SNNs. It is computationally simple but remains effective at capturing the dynamics of a biological neuron. The neuron receives incoming spikes from other neurons. The spikes are treated as input current I , causing the neuron’s membrane potential to increase.

This membrane potential undergoes decay over time, with rate $\beta \in (0, 1)$. Once a certain threshold value U_{thr} is reached, the neuron fires a spike, which is transmitted to adjacent neurons.

In our simulations, time steps are discrete and the LIF neurons’ membrane potential follow the equation :

$$U[t + 1] = \beta U[t] + I[t + 1] \quad (1)$$

Whenever $U[t] > U_{thr}$, $U[t + 1] = 0$ and the neuron is said to emit a spike at time $t + 1$, denoted as $\delta[t + 1] = 1$ ($\delta[t] = 0$ if no spike is emitted at time t).

B. Multilayered network

The neural architecture we use is a standard multilayered architecture where neurons are grouped by layers, and each neuron in a layer receives the outputs (here, the spikes) of all neurons in the previous layer. Neural connections are weighted, so that any neuron i that receives spikes from neurons j will have an input $I_i[t] = \sum_j w_{ij} \delta_j[t]$. See subsection IV-A for the details about the number and size of layers.

C. Gaussian Receptive Field (GRF) encoding

Gaussian receptive field (GRF) encoding, also known as population coding with overlapping Gaussian basis functions, is a biologically inspired method for representing continuous variables in neural systems. Rather than assigning a unique value to a single neuron, GRF encoding distributes the representation across a population of N neurons. Each neuron i is associated with a center μ_i within the input range and responds to input x according to a Gaussian function:

$$G_i(x) = \exp\left(-\frac{(x - \mu_i)^2}{2\sigma^2}\right) \quad (2)$$

where σ is the standard deviation, which determines the width of receptive fields. In this article, all receptive fields are assumed to have the same σ .

The centers $\{\mu_i\}$ are distributed evenly across the input domain. When an input value is presented, multiple neurons respond with various degrees of activation, resulting in an overlapping, distributed code, providing robustness to noise and allowing for smooth interpolation between encoded values.

D. Model

For the supervised learning of the synaptic weights, when dealing with a data sample (x, y) where x is the given feature vector and y the target value¹ for input x , we first apply a pre-defined transformation, or encoding function G , to the target value y . This produces a new target variable, $\gamma = \{G_i(y)\}_{i=1\dots N}$.

A SNN model is then trained to predict γ , thus generating an estimate, $\hat{\gamma}$, of the encoded target γ . Since models are not perfect and our model will undergo simulated hardware faults, this output is an approximation, so $\hat{\gamma} \approx \gamma$. We use the Mean Squared Error (MSE) between γ and $\hat{\gamma}$ as the loss function.

To obtain our final estimate for the original target value y , we apply a decoding function, f , to the model’s output $\hat{\gamma}$, thus $\hat{y} = f(\hat{\gamma})$. The model’s architecture is illustrated by Figure 1.

¹We assume here that $y \in \mathbb{R}$, but the method can be easily extended to a multidimensional target by using population coding for each output coordinate.

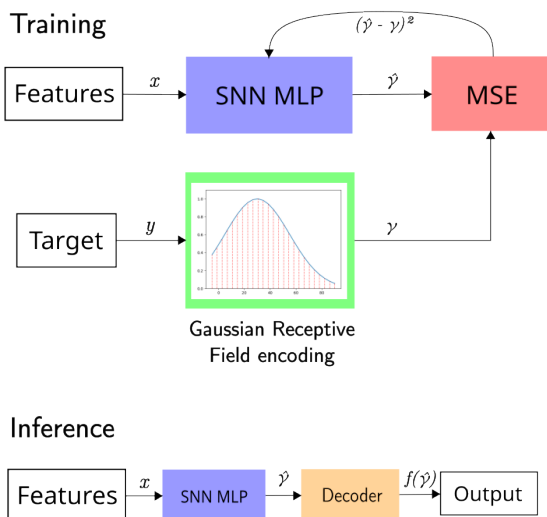


Fig. 1. The architecture of the GRF population model

Our goal is to find a decoder f such that \hat{y} is a close approximation of the true value y . The overall success of this method depends on both the model’s accuracy in predicting \hat{y} and the effectiveness of the decoder f in converting this prediction back to the original space, especially when dealing with noisy output and hardware faults.

E. Maximum Likelihood Estimation

In order to decode the value encoded by a GRF, we could use the Winner-Take-All (WTA) method which decodes the value by finding $\hat{y}_{WTA} = \mu_k$ where $k = \underset{i \in \{1, \dots, N\}}{\operatorname{argmax}}(\hat{\gamma}_i)$. Two apparent drawbacks of WTA are (1) this method can only be as accurate as the number of neurons N and (2) in the event of a “saturated neuron” fault in the output layer (see next subsection), the network will permanently predict the same value. In order to gain hardware fault tolerance, we propose the usage of the Maximum Likelihood Estimation (MLE) method. Intuitively, MLE enables the usage of multiple output neurons in the estimation, reducing the error induced by some faulty, possibly always firing, ones.

The underlying idea is that the output $\hat{\gamma}$ of the model should correspond to the population encoding of the decoded value \hat{y} . We thus try to find \hat{y} so that $G(\hat{y})$ is as close as possible to $\hat{\gamma}$. MLE assumes that the difference between $\hat{\gamma}$ and $G(\hat{y})$ looks like a gaussian noise:

$$\hat{\gamma} - G(\hat{y}) \simeq \{\mathcal{N}(0, \sigma')\}_{i \in \{1, \dots, N\}} \quad (3)$$

The likelihood function is then expressed as:

$$\mathcal{L}(\hat{y}, \hat{\gamma}) = \prod_{i=1}^N \exp\left(-\frac{(\hat{\gamma}_i - G_i(\hat{y}))^2}{2\sigma'^2}\right)$$

so that maximizing this likelihood is equivalent to minimizing the distance between $G(\hat{y})$ and $\hat{\gamma}$

$$\underset{\hat{y}}{\operatorname{argmax}} \mathcal{L}(\hat{y}, \hat{\gamma}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^N (\hat{\gamma}_i - G_i(\hat{y}))^2 \quad (4)$$

This minimization can then be achieved by any standard optimization algorithm (or even some kind of exhaustive search, since it would not involve a huge computational cost if reasonable absolute precision is expected).

F. Fault injection

Hardware faults may be permanent or transient. The most usual types of hardware faults are:

- **Stuck-at faults**, a data or control line appears to be held exclusively high (stuck-at-1) or low (stuck-at-0).
- **Random bit flips**, a data or memory element has some incorrect value.

The stuck-at fault model represents most permanent faults except those with indeterminate states. The random bit-flips model transient faults at the level of registers or memory elements, mostly due to external perturbations that affect the data (not the physical circuit itself). Based on similar considerations, [4] identifies different types of hardware faults for spiking neurons. They include “saturated neuron” fault, where the neuron is firing at every time-step and the “dead neuron” fault, where the neuron never fires, regardless of its membrane potential. For the sake of simplicity, we will only focus on those two kinds of faults.

G. Training

Since we do not study the impact of fault injection on the quality of on-chip training but only on the inference quality of pre-trained SNNs, the choice of a specific learning algorithm is not a central concern here. Yet, it is necessary to ensure that the pre-trained SNNs we use exhibit good performance so that their fault-free version stands as a relevant reference.

We chose to implement multilayered networks of fully connected Leaky Integrate and Fire (LIF) neurons, and to train them using Batch Normalization Through Time as it showed better performance in SNNs than regular Batch Normalization for static datasets [10]. We use a surrogate gradient derived from the \arctan function to compensate for the non-differentiability of the spiking function. We use rate coding for the output. Models were written in Pytorch [16] and snnTorch [5]. They were trained on an “RTX 2080 Ti” GPU. The chosen batch size was 64 for all models. The gradient-based learning algorithm, its learning rate, decay and the number of epochs were optimized using the Optuna framework [1] over 25 trials using 5-fold cross-validation, using 80% of the data for training and validation and 20% of the data as the held-out test set. Each model was trained using a learning rate scheduler and a tuned decay rate. The neurons have a threshold of 1 and a decay rate of 0.95. The d input features are scaled between 0 and 1, serving as constant input currents for the first layer of d neurons, over the 25 time steps of the simulation. The source code is available on https://github.com/agleyo/SNN_POPMLE.

We call “direct” the models that use a single neuron’s firing rate as the output and “population” the models that use the collective firing rates of a neuron population as the output, decoding them as described above. Direct models had a $[d, 256, 128, 1]$ architecture while population models had a $[d, 256, 128, N]$ architecture.

IV. EXPERIMENTAL RESULTS

A. Experimental setup

We used the concrete compressive strength dataset [25], the abalone dataset [15], the building energy efficiency dataset [24], the California housing dataset [9], the automobile miles per gallon (MPG) dataset [18] and the wine quality dataset [3] to test regression performances of the networks with or without injected faults. For each dataset, the centers of the receptive fields from the population network were set uniformly across the original training data interval, enlarged by 10% of its length on each side. This enlarged interval is meant to let bordering values be encoded by receptive fields coming from both left and right. The width σ of the receptive fields were determined according to the enlarged interval's length l , with the formula $\sigma = r_f \times l$, where r_f can be understood as the normalized receptive field size that is being tested.

After training, we injected a increasing ratio ρ of "dead neuron" and "saturated" faults in the networks and measured their Root Mean Square Error (RMSE). Each type of fault was injected in a separate experiment and was uniformly distributed across all neurons. Each model was trained 25 times using the hyperparameters found during the hyperparameter optimization phase. For each trained instance, models were injected with random faults and evaluated 50 times. The results are the average RMSE of the 25 training instances over the 50 fault injections.

The concrete compressive strength dataset was investigated more thoroughly than other datasets. We examined population sizes $N = 25$ and $N = 50$. We also tested different receptive field widths $r_f \in \{0.05, 0.10, 0.20\}$. Other datasets were tested with parameters $N = 50$ and $r_f = 0.10$.

B. Results

Regression results on the concrete compressive strength dataset are given in Figure 2 for the "dead neuron" fault and in Figure 3 for the "saturated neuron" fault.

Based on these results, different aspects can be outlined:

Saturated neurons vs dead neurons:

As illustrated by the scale of the RMSE axis in the two plots, "dead neuron" faults result in a less significant loss of performance than with "saturated neuron" faults, regardless of the model and fault rate.

Population (MLE) vs direct:

We found MLE population models to be significantly more robust to permanent fault injection than the direct networks, against both fault types.

Population (WTA) vs direct:

Although some WTA models had reasonable performance against the "dead neuron" faults, most of them had poor performance against the "saturated neuron" faults (see discussion below). One model with $N = 25$ and $r_f = 0.20$ stands out as having comparable performance with the MLE decoding method against "saturated neuron" faults but showed poor resilience against "dead neuron" faults.

Population size:

Dataset	$\rho = 0$	$\rho = 0.02$	$\rho = 0.06$	$\rho = 0.10$
Abalone (direct)	2.14	2.30	2.86	3.15
Abalone (population)	2.11	2.23	2.55	3.00
Energy (direct)	0.84	1.62	3.30	4.82
Energy (population)	0.55	0.68	0.95	1.21
Housing (direct)	0.61	0.66	0.77	0.88
Housing (population)	0.67	0.71	0.80	0.91
MPG (direct)	2.54	2.96	4.23	5.30
MPG (population)	2.46	2.5	2.63	2.79
Wine (direct)	0.64	0.72	0.87	0.99
Wine (population)	0.63	0.64	0.66	0.69

TABLE I
PREDICTION RMSE AFTER "DEAD NEURON" FAULT INJECTION

Dataset	$\rho = 0$	$\rho = 0.02$	$\rho = 0.06$	$\rho = 0.10$
Abalone (direct)	2.14	2.59	3.44	4.36
Abalone (population)	2.11	2.59	3.49	4.10
Energy (direct)	0.84	2.75	6.16	9.04
Energy (population)	0.55	1.25	3.13	5.18
Housing (direct)	0.61	0.79	1.1	1.28
Housing (population)	0.67	0.92	1.23	1.38
MPG (direct)	2.54	4.4	6.89	8.86
MPG (population)	2.46	3.06	4.31	5.29
Wine (direct)	0.64	1.27	1.69	1.87
Wine (population)	0.63	0.87	1.05	1.11

TABLE II
PREDICTION RMSE AFTER "SATURATED NEURON" FAULT INJECTION

There is no clear trend on how population size affects the resilience of tested MLE models. Receptive field width seems to be a more significant choice for this decoding method. However, population size did affect the resilience of WTA models. Contrary to what the inherent inaccuracy of having fewer possible output values would lead us to believe, networks with a smaller population size had higher resilience and base performance for WTA models.

Performances of direct and population on other datasets are given in Tables I and II. These results confirm that networks with population coding show more resilience to hardware faults than direct networks, despite contrasted results for different datasets. A similar resilience is obtained for the Abalone and Housing datasets (even slightly better for direct networks with the Housing dataset), whereas population networks outperform direct ones for all other datasets:

- For "saturated neurons" faults, a RMSE decrease between 30 % and 55 % is obtained thanks to population coding, depending on the dataset and fault rate.
- This RMSE decrease is always greater than the relative RMSE decrease obtained without injected faults thanks to population coding, meaning that population coding adds a stronger resilience to an initial better performance.
- For "dead neurons" faults, the same conclusions hold, though the RMSE decreases even more for the Energy dataset, but less for the other ones.
- The gain in resilience obviously increases with higher fault rates with these "dead neurons" faults. On average, the relative RMSE decrease with $\rho = 0.06$ is 60 % greater than with $\rho = 0.02$, and it is 80 % greater with $\rho = 0.10$.

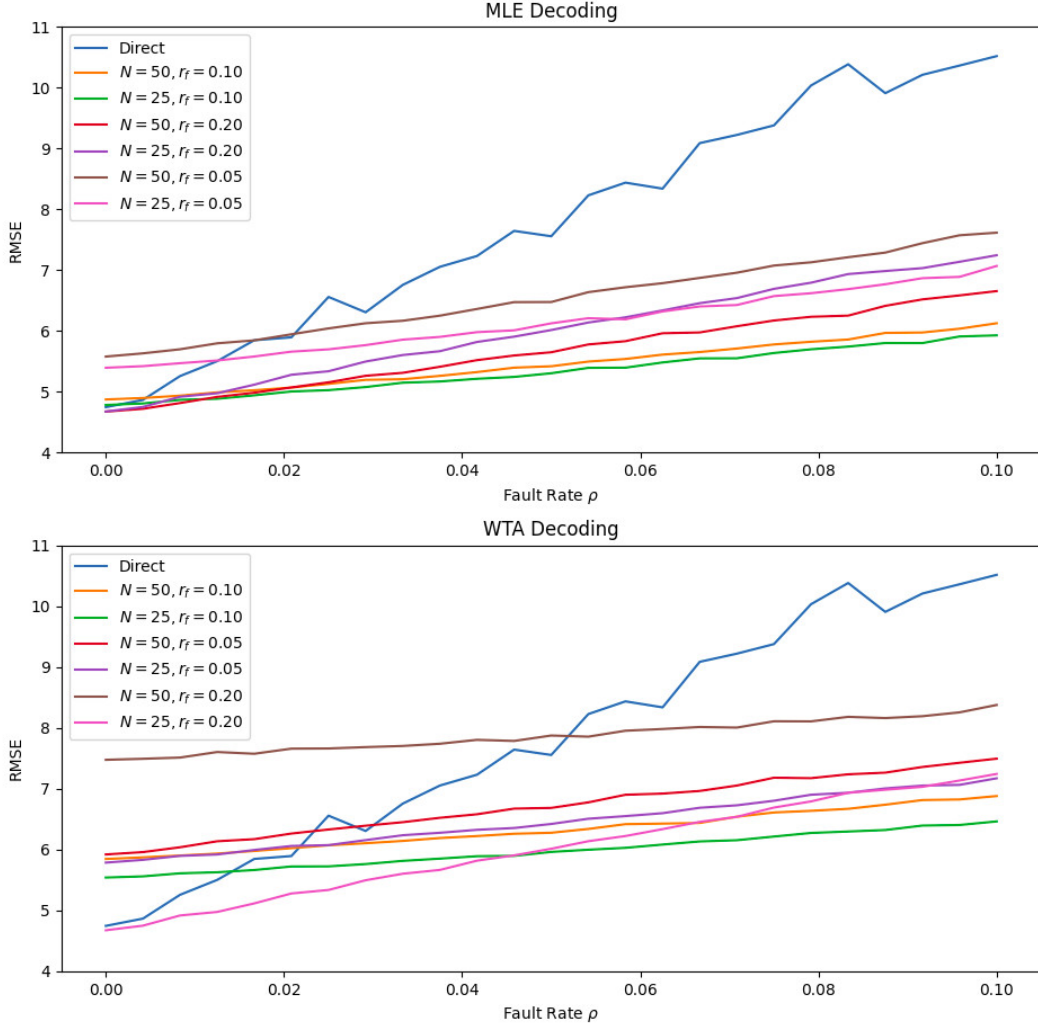


Fig. 2. Comparative fault tolerance of direct and population-coded networks under the “dead neuron” fault on the concrete compressive strength dataset. **(Top)** Population-coded models using the MLE decoding method: higher resilience than the direct network. The population network with $N = 25$ and $r_f = 0.10$ achieved the best results. **(Bottom)** Population-coded models using the WTA decoding method: higher resilience than the direct model but lower overall performance than the MLE-decoded models.

V. DISCUSSION

The WTA methods consistently showed less tolerance than their MLE counterparts, especially with “saturated neuron” faults. Both types of decoding performed better than the direct networks with “dead neuron” faults, but most WTA networks had worse performance than the direct network with “saturated neuron” faults. We can explain these results by the fact that a non-winning neuron in the output layer whose firing rate is lowered by a “dead neuron” fault does not affect the result of WTA, making the network resilient to this type of fault, while the distributed nature of the estimated receptive field for the MLE method implies that a highly stimulated neuron that has a lower-than-expected firing rate will shift the result to a neighboring value, generating only a small accuracy loss. Meanwhile, a neuron that has a higher-than-expected firing rate because of a “saturated neuron” fault might be selected as winning, decreasing the accuracy more significantly.

The MLE method’s interesting properties only appear when the sample size is large and the data is independent and identically distributed. Since N is both the size of the final layer of the network and the number of samples for the MLE, it must remain relatively small in comparison to the size of the network. Furthermore, the identical distribution property does not hold, especially after fault injection. Moreover, while the MLE method is applicable without fine-tuning, it might be outmatched by a Bayesian Model that is fine-tuned to the distribution of the training data. Despite the non-optimality of the MLE method, we believe we have proven the higher tolerance of population-coded spiking neural networks for both MLE and WTA decoding methods and the advantages of our MLE method on the WTA method.

While better performance of population models throughout different ratios of faults could partially be explained by a lower initial RMSE in the fault-free models for some datasets, we

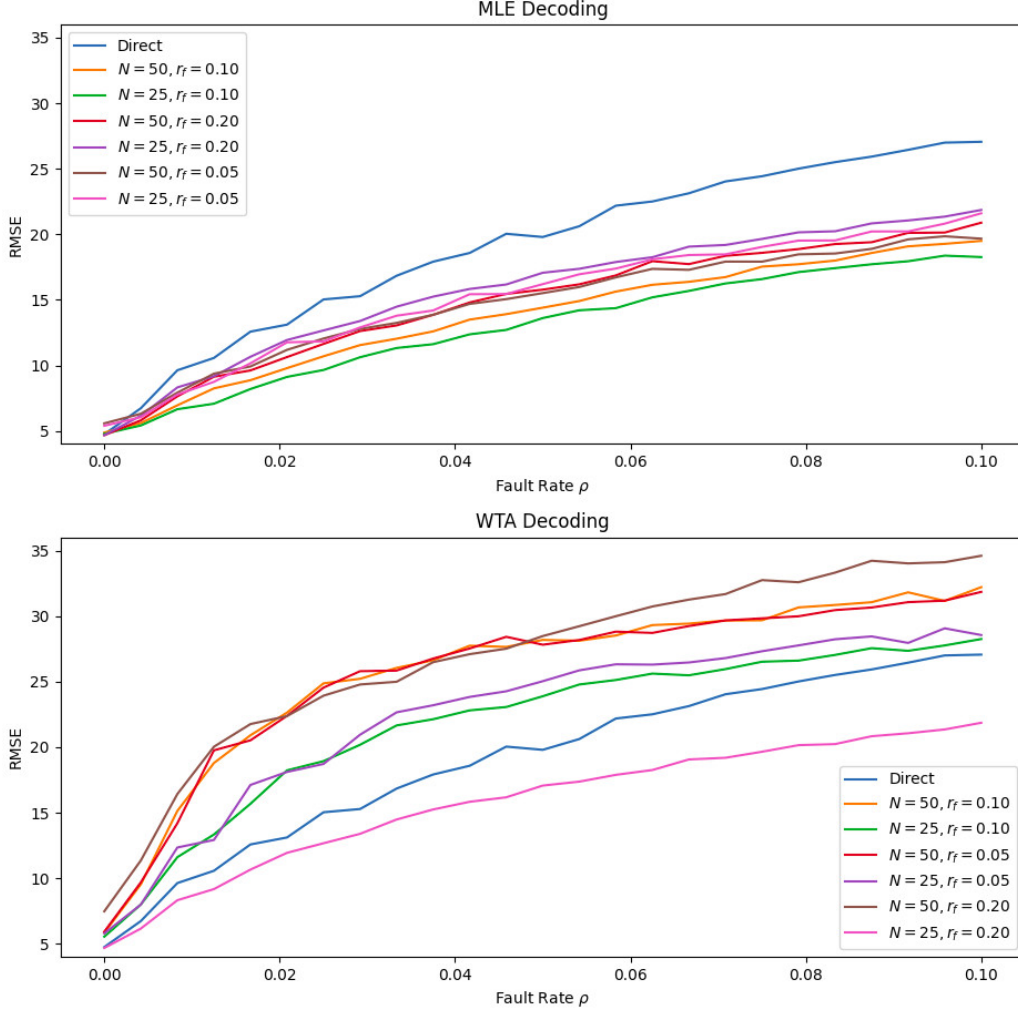


Fig. 3. Comparative fault tolerance of direct and population-coded networks under the "saturated neuron" fault on the concrete compressive strength dataset. **(Top)** Population-coded models using the MLE decoding method showed higher resilience than the direct network. The population network with $N = 25$ and $r_f = 0.10$ achieved the best results. **(Bottom)** All but one population-coded models using the WTA decoding method showed lower performance and lower resilience than the direct model.

observe in the concrete compressive strength experiments that even population networks that performed poorly in the absence of faults followed similar trends to other better-performing population networks, and ultimately outperforming the direct model for large values of ρ . Furthermore, both models had similar base performance for the Wine Quality dataset (0.64 vs 0.63) but had very different results for $\rho = 0.10$ (0.99 vs 0.69 for "dead neuron" faults and 1.87 vs 1.11 for "saturated neuron" faults). These results suggest that not only do population networks perform better regressions for some datasets, but also that their fault tolerance is due to the redundancy of the information they carry.

VI. CONCLUSION

In this paper, we investigated the inherent fault tolerance of population-coded Spiking Neural Networks for regression tasks. By testing and comparing several decoding methods as well as systemically injecting different types of faults, we have

shown that representing continuous variables with Gaussian Receptive Fields population codes provides significant resilience against both "dead" and "saturated" neuron faults, compared to the direct output model. Although the scope of this study is narrow since it only encompasses LIF neurons in relatively simple architectures, it provides a promising direction for future research.

Our network architecture requires no active fault detection and minimal hardware overhead. We have also found a Maximum Likelihood Estimation-based method that offers better resilience than the conventional Winner-Take-All method. This research validates that population coding is an efficient strategy for creating more reliable neuromorphic systems for deployment in resource-constrained environments. Future work should investigate more advanced decoding methods such as Bayesian inference.

REFERENCES

- [1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- [2] C. Alippi. Selecting accurate, robust, and minimal feedforward neural networks. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(12):1799–1810, 2002.
- [3] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009. Smart Business Networks: Concepts and Empirical Evidence.
- [4] Sarah A. El-Sayed, Theofilos Spyrou, Antonios Pavlidis, Engin Afacan, Luis A. Camuñas-Mesa, Bernabé Linares-Barranco, and Haralampos-G. Stratigopoulos. Spiking neuron hardware-level fault modeling. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–4, 2020.
- [5] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111(9):1016–1054, 2023.
- [6] Adrien Fois. *Plasticité et codage temporel dans les réseaux impulsionsnels appliqués à l'apprentissage de représentations*. PhD thesis, 2022. Thèse de doctorat dirigée par Girau, Bernard Informatique Université de Lorraine 2022.
- [7] Etash Guha, Shlok Natarajan, Thomas Möllenhoff, Mohammad Emtiyaz Khan, and Eugene Ndiaye. Conformal prediction via regression-as-classification, 2024.
- [8] Heiko Hoffmann. Advantages of neural population coding for deep learning, 2024.
- [9] R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.
- [10] Youngeun Kim and Priyadarshini Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch, 2021.
- [11] Edgar Lemaire, Loïc Cordone, Andrea Castagnetti, Pierre-Emmanuel Novac, Jonathan Courtois, and Benoît Miramond. *An Analytical Estimation of Spiking Neural Networks Energy Efficiency*, page 574–587. Springer International Publishing, 2023.
- [12] W. Maass. Noise as a resource for computation and learning in networks of spiking neurons. *Proceedings of the IEEE*, 102(5):860–880, 2014.
- [13] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.
- [14] Sparsh Mittal. Soft errors in dnn accelerators: A comprehensive review. *Microelectronics Reliability*, 115:113969, 2020.
- [15] Warwick Nash, Tracy Sellers, Simon Talbot, Andrew Cawthorn, and Wes Ford. Abalone. UCI Machine Learning Repository, 1994. DOI: <https://doi.org/10.24432/C55C7W>.
- [16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.
- [17] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Rescuesnn: enabling reliable executions on spiking neural network accelerators under permanent faults. *Frontiers in Neuroscience*, 17, April 2023.
- [18] R. Quinlan. Auto MPG. UCI Machine Learning Repository, 1993. DOI: <https://doi.org/10.24432/C5859H>.
- [19] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. Methods for fault tolerance in networks-on-chip. *ACM Comput. Surv.*, 46(1), July 2013.
- [20] C. David Schuman et al. Resilience and robustness of spiking neural networks for neuromorphic systems. *IJCNN*, 2020. Revue sur la résilience et les limites de tolérance intrinsèque.
- [21] Terry Sejnowski and Tobi Delbruck. The language of the brain. *Scientific American*, 307:54–59, 2012.
- [22] Theofilos Spyrou, Sarah A. El-Sayed, Engin Afacan, Luis A. Camuñas Mesa, Bernabé Linares-Barranco, and Haralampos-G. Stratigopoulos. Reliability analysis of a spiking neural network hardware accelerator. In *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*, 2022. Injection de fautes matérielles sur accélérateur FPGA SNN.
- [23] Tao Sun and Sander Bohté. Average-over-time spiking neural networks for uncertainty estimation in regression, 2024.
- [24] Athanasios Tsanas and Angeliki Xifara. Energy Efficiency. UCI Machine Learning Repository, 2012. DOI: <https://doi.org/10.24432/C51307>.
- [25] I-Cheng Yeh. Concrete Compressive Strength. UCI Machine Learning Repository, 1998. DOI: <https://doi.org/10.24432/C5PK67>.