New Wine in an Old Bottle: N-Version Programming for Machine Learning Components

Arpan Gujarati
Max Planck Institute for Software Systems
arpanbg@mpi-sws.org

Sathish Gopalakrishnan, Karthik Pattabiraman
The University of British Columbia
{sathish,karthikp}@ece.ubc.ca

***Abstract*—We revisit N-version programming in the context of machine learning (ML). Generating N versions of an ML component does not require additional programming effort, but only extra computations. This opens up the possibility of executing hundreds of diverse replicas, which, if carefully deployed, can improve their overall reliability by a significant margin. We use mathematical modeling to evaluate these benefits.**

I. Introduction

Driven by breakthroughs in machine learning (ML) and the widespread availability of ML frameworks, cyber-physical systems (CPS) that interact intelligently with the world are becoming part of our everyday lives. They range from smart devices in the home, to retail and restaurant robots, to self-driving cars and self-flying drones. However, the extensive use of ML for automation entails significant reliability and safety risks. For example, in 2018, an Uber self-driving car operating in autonomous mode struck and killed a pedestrian.

In general, at least two challenges need to be addressed before we can trust the use of ML in safety-critical CPS: **(i)** improving the baseline accuracy of ML algorithms, and **(ii)** reducing the impact of software bugs and hardware faults, which is exacerbated by the use of off-the-shelf ML frameworks on commodity platforms. *Our focus is on the latter challenge.*

The classical approach to tolerating both software bugs and hardware (design) faults is *N-version programming* (NVP) [1]. Central to NVP is the idea that independently generated *programmed components* are diverse and thereby reduce the probability of identical bugs and faults. However, prior work has argued that failures in N-version programs are not statistically independent [2, 3], and that the time and costs of developing a complete N-version execution environment (NVX) is often not worthwhile in practice [4].

Our goal here is to revisit NVP in the context of *ML components*, which are not programmed but trained, using supervised, unsupervised or reinforcement learning. Our hypothesis is that NVP for ML components is actually like *new wine in an old bottle*, *i.e.*, although the basic principles and terminology *viz.* "NVP" and "NVX" remain the same, the exact mechanisms and costs involved in generating and deploying N different versions of an ML component are different.

Generating N functionally identical and yet diverse ML components does not require extra programming effort; this requires additional computations [5, 6]. For instance, consider deep neural networks (DNNs): different ML frameworks such as PyTorch, TensorFlow, and TVM can be used to generate ML models with different execution plans; DNNs can be trained with different network structures (*e.g.*, image recognition using ResNet50, ResNet101, and ResNet152); and *ensemble techniques* [7] could be used to train ML models using distinct random choices. These options open up the possibility of generating and executing hundreds of diverse replicas inside an NVX, which was impossible before.

In addition, unlike programmed components, whose baseline reliability is already high (and typically measured in "nines"), the baseline reliability of ML component is relatively low (*e.g.*, an inference accuracy of 75% to 90% is common among DNNs). NVP therefore has a huge potential to improve the overall reliability of ML components. Hence, we believe there is a need to investigate the problem of NVP for ML components with a fresh perspective, and also explore different NVX configurations. In this regard, we present an initial study assessing the benefits of NVP for ML components in the presence of permanent faults. Our main findings are as follows:

- *Concurrent* execution of replicas can improve the reliability of ML components by a significant margin, but only if the number of faults is not very high.
- *Sequential* execution is ineffective for low diversity values, but if the diversity percentage among each replica pair is increased (say, beyond 75%), it can be as effective as concurrent execution in providing high reliability.
- The voting algorithm plays an important role. *Simple majority* voting can reverse the reliability gains that are otherwise achievable using more lenient quorum sizes.

Another interesting takeaway from our work is that NVP can be used to improve the baseline accuracy of ML components as well, and not just their reliability in the presence of faults. This indicates that NVP has the potential to kill two birds with one stone, and merits a thorough investigation going forward.

II. Reliability Modeling

Assumptions and axiomatic properties. We assume that an ML component refers to a trained DNN model, and hence we use these terms interchangeably. We also consider only permanent faults in this study. Our reliability modeling is based on three axiomatic properties as follows.

[1]This work was supported in part by research grants from the Natural Sciences and Research Council of Canada (NSERC), Huawei, and the Peter Wall Institute for Advanced Studies at The University of British Columbia.

1) *A trained DNN*, even in the absence of any faults, may not always output the correct answer. Thus, we consider its *baseline reliability*, often referred to as its *acccuracy*, to be anywhere in the interval $[0, 1)$.
2) Permanent faults may interfere with the functioning of individual hardware units. We also assume that *the higher the number of faults, higher is the probability that the DNN outputs a wrong inference.*
3) This property pertains to the training of architecturally diverse DNN models for the same functional objective, such as image detection. *As a result, depending on the degree of diversity, a fault may trigger correlated failures in two diverse replicas or only independent failures.*

Modeling axiomatic properties. Based on prior work by Zhang et al. [8], we use an exponential function to approximate a DNN's reliability $R(x)$ in the presence of x faults:

$$R(x) = \alpha e^{-\beta x} \ (\textit{where } \alpha < 1). \tag{1}$$

Equation (1) sufficiently models the first two axiomatic properties, since the baseline reliability is less than one, *i.e.*, $R(0) = \alpha < 1$, and since the reliability decreases with increasing number of faults, *i.e.*, $R(y) < R(x)$ for $y > x$.

There is a paucity of empirical data on how the reliabilities of diverse ML components, *e.g.*, DNNs generated using different ML frameworks, are related. Therefore, to model the third axiomatic property, we make a simple modeling assumption, namely that each ML component replica is split logically into two ML subcomponents (see Figure 1(a)). We refer to these as the *identity* and *diversity* subcomponents. While the identity subcomponent remains same across replicas, the diversity subcomponent varies from one replica to another.

Given a replica, let $R_{1,a}(x)$ and $R_{1,b}(x)$ denote the reliability of its identity and diversity subcomponents. We define the replica reliability $R_1(x)$ as a *weighted geometric mean* of $R_{1,a}(x)$ and $R_{1,b}(x)$. Given respective weights w_a and w_b,

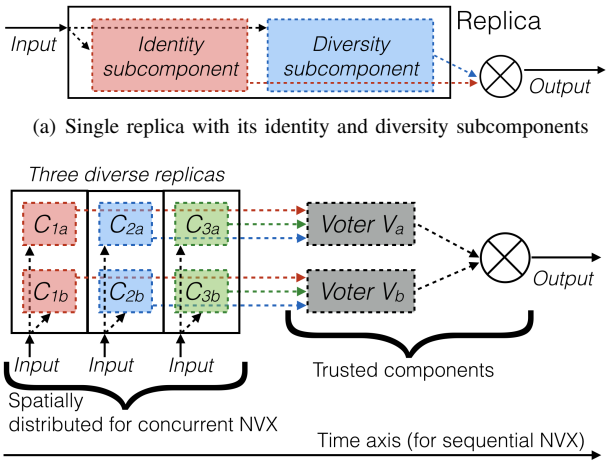
$$\begin{aligned} R_1(x) &= \mathit{WeightedGM}\left(R_{1,a}(x), R_{1,b}(x), w_a, w_b\right) \\ &= \left(R_{1,a}(x)^{w_a} \times R_{1,b}(x)^{w_b}\right)^{\frac{1}{w_a+w_b}}, \end{aligned} \tag{2}$$

where $d = 100w_b/(w_a + w_b)$ denotes intuitively the diversity percentage among all ML component replicas.

The models introduced above provide us with a set of useful tools to approximate the reliability of different NVX configurations. In the following, we explore the reliability of *sequential* and *concurrent* NVX configurations.

Modeling sequential execution. In the sequential execution model, the replicas execute one after another on a single shared hardware platform, followed by voting (or some form of merging) to suppress the redundant outputs. Building up on the models introduced in the aforementioned paragraphs, we first decompose the execution sequence into sequential executions of identity and diversity subcomponents of each replica.

We assume that outputs of all identity subcomponents $C_{1,a}$, $C_{2,a}$, …, $C_{N,a}$ are first passed through voter V_a (as illustrated in Figure 1(b)). Similarly, outputs of all diversity subcomponents $C_{1,b}$, $C_{2,b}$, …, $C_{N,b}$ are then passed through



(a) Single replica with its identity and diversity subcomponents

(b) Three replicas in sequential and concurrent NVX configurations

Fig. 1. NVX with replica diversity (colors used only to separate components).

voter V_b. Inputs to any component is independent of the outputs of previously executed components (*e.g.*, input to $C_{3,a}$ is independent of $C_{1,a}$). In the final step, outputs of voters V_a and V_b are composed to generate the NVX output. The last step is similar to the output composition shown in Figure 1(a).

Let $R_{NVX,seq}(x)$ denote the sequential NVX reliability in the presence of x faults. $R_{NVX,seq}(x)$ can be computed in a straightforward manner, like in Equation (2) using a weighted geometric mean, if the reliabilities of voters V_a and V_b's outputs are known. Let these reliability be denoted as $R_{NVX,seq,a}(x)$ and $R_{NVX,seq,b}(x)$, respectively.

Computing $R_{NVX,seq,a}(x)$ is trivial since the inputs to voter V_a come from identity subcomponents $C_{1,a}$, $C_{2,a}$, …, $C_{N,a}$, which are expected to fail identically in the presence of permanent faults. In other words, voting is irrelevant here, because either all identity subcomponents fail, or none of them fail (we ignore the small chance that a permanent fault occurs in the middle of a sequential execution, thereby invalidating Equation (3)). Thus, we have the following equalities,

$$R_{1,a}(x) = R_{2,a}(x) = \ldots = R_{n,a}(x) = R_a(x) \tag{3}$$

$$R_{NVX,seq,a}(x) = R_a(x). \tag{4}$$

In contrast, computing $R_{NVX,seq,b}(x)$ is not trivial since the inputs to voter V_b come from diversity subcomponents $C_{1,b}$, $C_{2,b}$, …, $C_{N,b}$, which fail differently in the worst case. Therefore, we enumerate all possible scenarios, which accounts for the possibility of each subcomponent $C_{i,b}$ being affected by faults and not affected by faults, *i.e.*, up to 2^N scenarios. Assuming that voter V_b requires a minimum quorum size of $q \leq N$, and letting $\mathbb{N}_N = \{1, 2, \ldots, N\}$, we define

$$R_{NVX,seq,b}(x) = \sum_{i=q}^{N} \sum_{\substack{S \subseteq \mathbb{N}_N \\ \& \ |S|=i}} \left(\prod_{j \in S} R_{j,b}(x) \right) \left(\prod_{j \in \mathbb{N}_N \setminus S} (1 - R_{j,b}(x)) \right). \tag{5}$$

In Equation (5), the first summation from the left (*i.e.*, from $i = q$ to $i = n$) enumerates scenarios with quorum sizes greater

than or equal to the minimum quorum size q. Given a quorum size i, the second summation enumerates over all permutations of $N - i$ faulty replicas, *i.e.*, in each permutation, i out of N subcomponents yield correct answers, whereas others are assumed to yield faulty answers. The first product from the left computes the combined reliability of all correctly executed subcomponents. The second product computes the combined failure probability of all the faulty subcomponents.

Next, we compose the reliabilities of voters V_a and V_b's outputs, like in Equation (1). This yields the reliability of a sequential NVX in the presence of x permanent faults.

$$R_{NVX,seq}(x) = WeightedGM \begin{pmatrix} R_{NVX,seq,a}(x), \\ R_{NVX,seq,b}(x), \\ w_a, w_b \end{pmatrix}. \quad (6)$$

Modeling concurrent executions. In the concurrent execution model, replicas execute concurrently on independent hardware platforms. The voting step is similar to that in a sequential execution. While the objective of concurrent execution is typically to tolerate crash failures, it also helps to minimize the chances of correlations among faulty replica outputs (it is highly unlikely that replicas are exposed to the same set of faults on independent hardware platforms).

The reliability model for a concurrent NVX can be derived assuming that identity subcomponents are affected by the faults differently. That is, while Equation (3) still holds for all identity subcomponents, *i.e.*, $R_{1,a}(x) = R_{2,a}(x) = \ldots = R_{N,a}(x) = R_a(x)$, we define voter V_a's reliability $R_{NVX,conc,a}(x)$ differently from Equation (4), as follows:

$$\begin{aligned} &R_{NVX,conc,a}(x) \\ &= \sum_{i=q}^{N} \sum_{\substack{S \subseteq \mathbb{N}_N \\ \& \ |S|=i}} \left(\prod_{j \in S} R_{j,a}(x) \right) \left(\prod_{j \in \mathbb{N}_n \setminus S} (1 - R_{j,a}(x)) \right) \\ &= \sum_{i=q}^{N} \binom{N}{i} R_a(x)^i (1 - R_a(x))^{N-i} \ \{\textit{using Equation (3)}\}. \quad (7) \end{aligned}$$

Voter V_b's reliability definition and the overall NVX reliability remain the same as in for sequential NVX. That is,

$$\begin{aligned} &R_{NVX,conc,b}(x) = \\ &\sum_{i=q}^{N} \sum_{\substack{S \subseteq \mathbb{N}_N \\ \& \ |S|=i}} \left(\prod_{j \in S} R_{j,b}(x) \right) \left(\prod_{j \in \mathbb{N}_N \setminus S} (1 - R_{j,b}(x)) \right), \quad (8) \end{aligned}$$

$$\begin{aligned} &\textit{and } R_{NVX,conc}(x) = \\ &WeightedGM \begin{pmatrix} R_{NVX,conc,a}(x), R_{NVX,conc,b}(x), \\ w_a, w_b \end{pmatrix}. \quad (9) \end{aligned}$$

III. Evaluation

The objective of the numerical evaluation is to derive actionable insights using the proposed reliability modeling through design-space exploration. We defer the problem of validating the models themselves through simulation or through fault injection to future work.

We assume throughout that each $R_{i,a}(x) = R_{j,b}(x) = R(x)$, where $R(x)$ is defined as in Equation (1). We then use non-linear least squares to fit $R(x)$ to the empirical data provided by Zhang *et al.* [8] where they evaluate the accuracy of MNIST digit classification and TIMIT speech recognition tasks with respect to the number of faulty Multiply-Accumulate (MAC) units in a DNN accelerator. The data and the values obtained for parameters α and β through curve fitting are shown below (x denotes the number of faulty MACs). Note that our model is applicable to other types of hardware and software faults as well. Unfortunately, there has been little published work, especially on the effect of software faults on ML components.

ML task	Accuracy % for different values of x [8]								Fitted params	
	0	1	2	4	8	16	32	64	α	β
TIMIT	74	70	70	40	46	4	2	2	77.4	0.11
MNIST	98	94	96	82	64	48	20	12	99.4	0.05

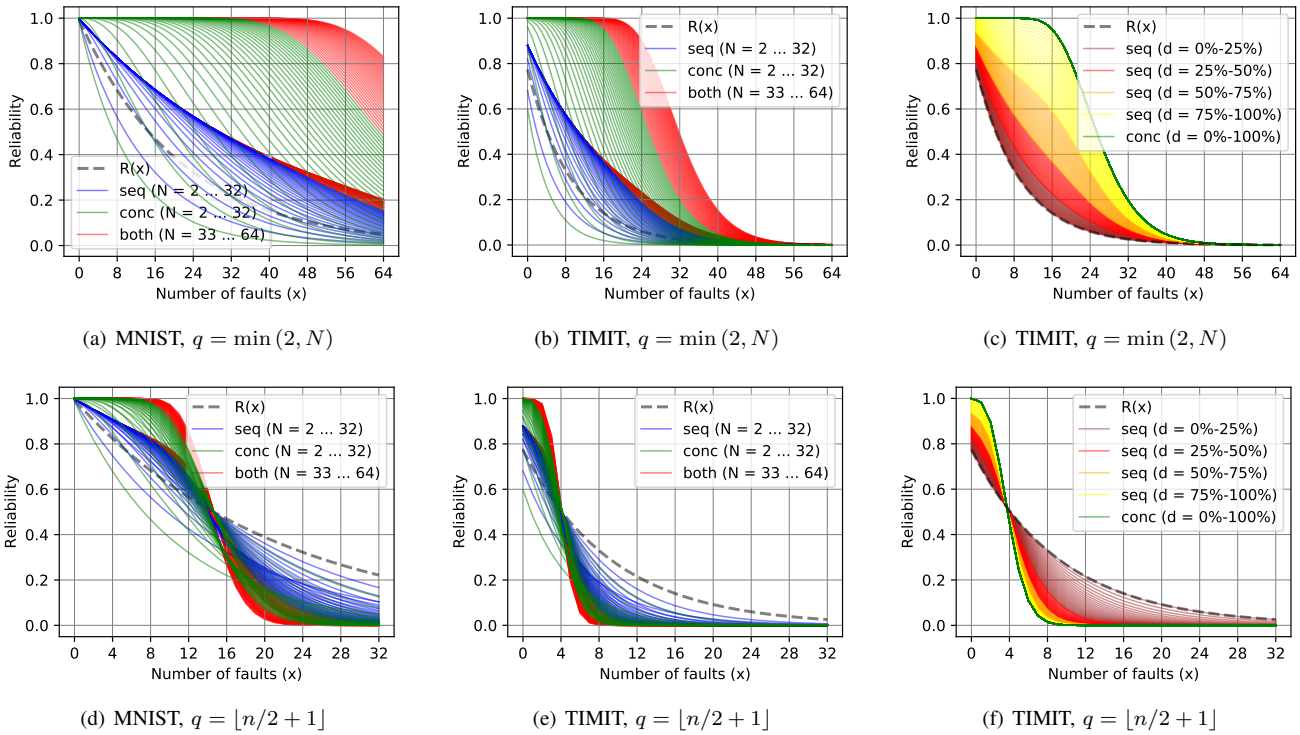
Overall trends. To understand the general trends, we first computed the reliability for sequential and concurrent NVX using Equations (6) and (9) for two different configurations. **(i)** In the first case, function $R(x)$ is fitted to the MNIST data, the minimum quorum size is $q = \min(2, N)$, diversity percentage $d = 100w_b/(w_a + w_b) = 50\%$, and the number of replicas N is varied from 2 to 64. **(ii)** In the second case, the only change is that $R(x)$ is fitted to the TIMIT data. Results are illustrated in Figures 2(a) and 2(b), respectively.

We observe that more replication always helps, since reliability values for $N \in \{33, \ldots, 64\}$ are strictly higher than those for $N \in \{2, \ldots, 32\}$ for respective execution types. Secondly, concurrent executions are generally more reliable than sequential executions, at least as long as the number of faults x is smaller than some threshold x_{max}, beyond which many sequential executions start offering relatively higher reliability. Threshold x_{max} depends on both $R(x)$ and N.

Most importantly, Figures 2(a) and 2(b) show that concurrent NVX can improve the reliability of ML components by a significant margin. In the case of TIMIT, the overall NVX reliability can get very close to 1 despite its baseline reliability being $R(0) = 74\%$. This suggests that it is possible to increase the reliability of the system by generating a large number of replicas with very high diversity (50%) among each pair.

Effect of quorum size. In Figure 2(c), we illustrate results for the TIMIT data once again with a minimum quorum size of $q = \min(2, N)$; but unlike in Figure 2(b), we fix the number of replicas at $N = 32$ and instead vary the diversity percentage $d = 100w_b/(w_a + w_b)$ from 0% to 100%. As expected, the reliability of concurrent NVX is independent of d, since we assumed that faults on independent hardware platforms do not overlap. On the other hand, the reliability of sequential NVX approaches $R(x)$ at $d = 0$ and the reliability of concurrent NVX at $d = 100$. Thus, by introducing sufficient diversity, even sequential NVX can offer significant reliability.

Putting it all together. In this last numerical study, we evaluated all configurations discussed above but with a mini-



(a) MNIST, $q = \min(2, N)$ (b) TIMIT, $q = \min(2, N)$ (c) TIMIT, $q = \min(2, N)$

(d) MNIST, $q = \lfloor n/2 + 1 \rfloor$ (e) TIMIT, $q = \lfloor n/2 + 1 \rfloor$ (f) TIMIT, $q = \lfloor n/2 + 1 \rfloor$

Fig. 2. The dataset, *i.e.*, MNIST or TIMIT, and the minimum quorum size q are stated for each figure separately. X axis denotes the number of faults x and Y axis denotes the reliability (a probability between 0 and 1). $R(x)$ is the baseline reliability obtained using Equation (1) and denoted using a grey dashed curve. In insets (a), (b), (d) and (e), the number of replicas N is varied from 2 to 64 but the diversity percentage d is fixed at 50%. The blue and green curves in these figures indicate reliability values for sequential and concurrent NVX (respectively), for $N = \{2, \ldots, 32\}$. The red curves on the other hand indicate results for both sequential and concurrent NVX but for $N = \{33, \ldots, 64\}$, *i.e.*, red curves indicate more replication. In insets (c) and (f), the number of replicas is fixed at $N = 32$ but the diversity percentage d is varied from 0% to 100%. For concurrent NVX, the reliability is denoted using green color, and all curves overlap. For sequential NVX, as the diversity percentage increases from 0% to 100%, the color of the reliability curves change from brown to yellow.

mum quorum size of $q = \lfloor n/2 + 1 \rfloor$ (simple majority) instead of $q = \min(2, N)$. Results are illustrated in Figures 2(d) to 2(f). With a larger quorum size, more replication helps only up to a certain value of x, *e.g.*, in Figure 2(e), after $x = 4$, the reliability for every configuration is less than $R(x)$. The reliability gains offered by different configurations are much smaller. This is typical of majority-voting systems where the minimum quorum size is proportional to N; after some faults, a quorum of correct values becomes highly improbable.

IV. Conclusion and Future Work

NVP has faced criticism for increasing the reliability traditional (programmed) software components [2, 3]. However, driven by the observation that NVP for ML components is vastly different, we revisited this technique for such components. In the case of MNIST digit classification and TIMIT speech recognition tasks, our mathematical modeling and experiments showed that both concurrent and sequential NVX can improve the overall reliability by a significant margin if a large number of high diversity replicas can be generated. In general, there are immense benefits of using NVP for ML components. Especially, the possibility of increasing the baseline reliability (accuracy) of many ML components in the presence of faults is encouraging, as also observed by some recent works [9, 10].

In the near future, we plan to explore three broad directions of work. First, on the ML side, specifically for DNNs, we plan to work on quantifying the diversity of replicas with respect to the input space, network structure, and weights, which in turn translates into diversity in the presence of inputs and software/hardware faults. Second, we require high-fidelity simulations or a thorough fault injection to validate the proposed models, *i.e.*, does the logical decomposition into identity and diversity subcomponents match empirical results? We also intend to work on design of such systems, and a key aspect is the design of voting mechanisms for ML components.

References

[1] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *IEEE FTCS-8*, 1978.

[2] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on Software Engineering*, no. 1, pp. 96–109, 1986.

[3] ——, "A reply to the criticisms of the Knight & Leveson experiment," *ACM SIGSOFT Software Engineering Notes*, vol. 15, no. 1, pp. 24–35, 1990.

[4] L. Sha *et al.*, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[5] S. Latifi, B. Zamirai, and S. Mahlke, "PolygraphMR: Enhancing the reliability and dependability of CNNs," in *IEEE DSN*, 2020.

[6] R. Salay, R. Queiroz, and K. Czarnecki, "An analysis of ISO 26262: Machine learning and safety in automotive software," in *SAE Technical Paper*, 2018.

[7] M. P. Ponti Jr, "Combining classifiers: from the creation of ensembles to the decision fusion," in *IEEE SIBGRAPI*, 2011.

[8] J. J. Zhang, K. Basu, and S. Garg, "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Design & Test*, vol. 36, no. 5, pp. 44–53, 2019.

[9] H. Xu, Z. Chen, W. Wu, Z. Jin, S.-y. Kuo, and M. Lyu, "NV-DNN: towards fault-tolerant DNN systems with N-version programming," in *IEEE/IFIP DSN-W*, 2019.

[10] F. Machida, "N-version machine learning models for safety critical systems," in *IEEE/IFIP DSN-W*, 2019.