2023 28th IEEE European Test Symposium (ETS)

Online Performance Monitoring of Neuromorphic Computing Systems

Abhishek Kumar Mishra
ECE Department
Drexel University
Philadelphia, USA
am4862@drexel.edu

Anup Das
ECE Department
Drexel University
Philadelphia, USA
anup.das@drexel.edu

Nagarajan Kandasamy
ECE Department
Drexel University
Philadelphia, USA
kandasamy@drexel.edu

2023 IEEE European Test Symposium (ETS) | 979-8-3503-3634-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ETS56758.2023.10173860

***Abstract*—Neuromorphic computation is based on spike trains in which the location and frequency of spikes occurring within the network guide the execution. This paper develops a framework to monitor the correctness of a neuromorphic program's execution using model-based redundancy in which a software-based monitor compares discrepancies between the behavior of neurons mapped to hardware and that predicted by a corresponding mathematical model in real time. Our approach reduces the hardware overhead needed to support the monitoring infrastructure and minimizes intrusion on the executing application. Fault-injection experiments utilizing CARLSim, a high-fidelity SNN simulator, show that the framework achieves high fault coverage using parsimonious models which can operate with low computational overhead in real time.**

***Index Terms*—Neuromorphic computing, spiking neural networks, model-based fault detection, online monitoring**

I. Introduction

Neuromorphic platforms execute *SNNs* which are networks of spiking neurons interconnected via synapses in which neurons communicate with each other by sending short impulses of infinitesimally small duration, called *spikes*, via synapses. Such spiking neurons can be organized into *feed-forward layers* or in a *recurrent topology*. A typical feed-forward SNN has one input layer, one or more hidden layers, and one output layer. Other widely-used neural network models, such as convolutional neural networks (CNN), can also be systematically converted to equivalent SNNs and mapped to neuromorphic hardware [1]. These can be deployed in mobile devices and native sensors, which have low-power requirements and must operate in real time.

SNNs are typically used to solve classification problems in image and signal processing. The computation performed within the SNN is based on asynchronously occurring *spike trains* in which the location and frequency of spikes occurring within the network guide the execution. For example, the *inter-spike interval* (*ISI*) or the time between subsequent spikes encodes information critical to the correctness of the computation — which in this case is classification accuracy.

The work described in this paper concerns *dependability of neuromorphic computing* — specifically, monitoring the SNN's execution on the underlying neuromorphic platform for correctness in the face of hardware failures. We provide answers to the following questions: (1) how do we determine correctness of an SNN executing on neuromorphic hardware; (2) what are the appropriate metrics that can detect manifestation of faults in this hardware; (3) what is the hardware/software interface needed to collect these metrics; and (4) how to develop a low-cost monitoring framework to assess the health of neuromorphic hardware.

- We describe the design of *performance monitoring units* (*PMUs*) to gather key statistics on synaptic events occurring within the neuromorphic hardware, which are spike rate and the average ISI for a neuron. Counters within the PMU accumulate these events in real time and make them accessible to the monitor. The PMUs are designed to minimize intrusion on the executing SNN.
- We develop a *model-based approach for fault detection and isolation* (*FDI*) in which software-based monitors compare discrepancies between the behavior of neurons mapped to hardware to that predicted by corresponding mathematical models in real time.
- Missing or delayed or spurious spikes from neurons within the SNN, which are the manifestation of faults affecting the crossbars, tend to distort the ISI, resulting in performance loss. Therefore, we develop *parsimonious models*, suitable for use in real time, to estimate ISI while minimizing the number of such models needed for full coverage of crossbars within the neuromorphic hardware.
- We use CARLsim, a library which simulates SNNs [2], to generate synthetic networks and inject faults which cause ISI distortion in spike trains. Our approach is evaluated in terms of fault-detection rate as well as computational and storage overhead incurred by the models.

II. Background

A crossbar is a two-dimensional arrangement of synapses, with n^2 synapses for n input neurons. To build large neuromorphic hardware, the common practice is to integrate multiple such crossbars along with a shared interconnect [3]. This hardware abstraction fits most neuromorphic architectures and will be used in this paper.

Figure 1 shows the 3D-view of a crossbar in terms of top electrodes (TEs) which form rows and bottom electrodes (BEs) which form columns [4]. A synaptic cell is connected at a crosspoint via an access transistor. Synaptic weights are specified in terms of conductivity of non-volatile memory (NVM) cells, allowing these cells to act as computational units through analog summation of the current that flows through them [5]. The NVM is shown as a resistive element in Fig. 1. Pre-synaptic neurons are mapped along the TEs and post-synaptic neurons along the BEs. The synaptic weight between a pre- and a post-synaptic neuron is programmed as conductance of the corresponding synaptic cell at the crosspoint. A pre-synaptic neuron's voltage v, applied on the TE, is multiplied by the conductance to generate current according to Ohm's Law. Current summation occurs on each BE according to Kirchoff's Current Law, when integrating excitation from other pre-synaptic neurons. The figure shows the integration of input excitation from two pre-synaptic neurons to one post-synaptic neuron via synaptic weights w_1 and

979-8-3503-3634-4/23/$31.00 ©2023 IEEE

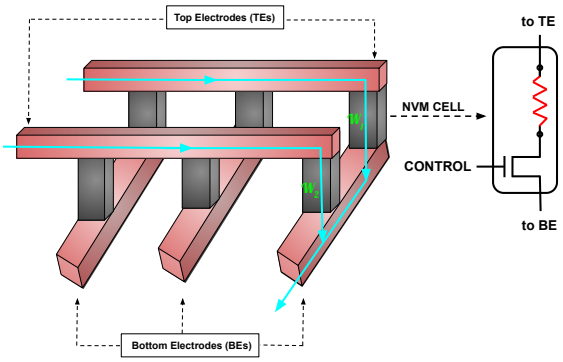
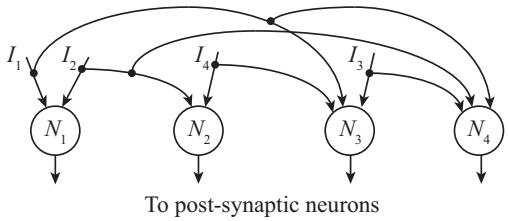
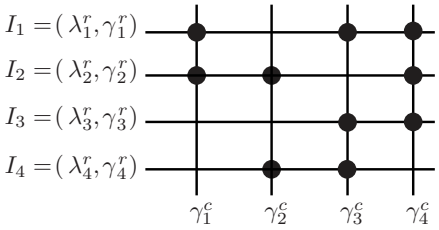


Fig. 1: Crossbar organization showing the top and bottom electrodes. Each synaptic cell consists of an NVM device (resistive element) and an access transistor.



(a) An SNN consisting of four neurons.



(b) Neurons mapped to a 4×4 crossbar.

Fig. 2: A small SNN mapped to a 4×4 crossbar.

w_2, respectively. As per Kirchhoff's law, current summation along the column implements the sum $w_1 v_1 + w_2 v_2$ needed for forward propagation of neuron activation. These current summations are performed along each column in parallel.

The NVM device of a synaptic cell can be implemented using technologies such as phase-change memory or oxide-based memory [5]. To read or to program a cell, its peripheral circuit drives current through it using a bias voltage generated by on-chip charge pumps built using CMOS devices. This voltage must be high enough to compensate for IR drop and the built-in potential of the access device, which connects the cell to a row and a column in the crossbar.

SNNs are networks of spiking neurons interconnected via synapses. These neurons are typically implemented using *integrate-and-fire* models in which the membrane voltage of a neuron increases due to current at its input from other neurons [6]. A neuron fires a spike when its membrane voltage exceeds a threshold and subsequently the membrane voltage is reset. The refractory period refers to a brief rest period after a spike is fired, during which the neuron cannot fire again.

SNNs have been widely researched in recent years for use in spatio-temporal pattern recognition applications. To train an SNN, synaptic weights are adjusted using supervised, semi-supervised, or unsupervised learning [7]. To map an SNN to hardware, we start its software model or a previously trained CNN which has been converted to an equivalent SNN. A synapse-to-crossbar as well as crossbar-to-charge pump mapping is generated for the specific neuromorphic hardware

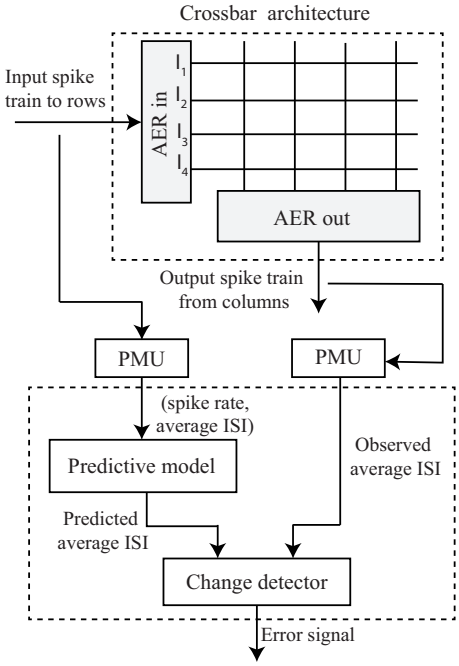


Fig. 3: Model-based performance monitoring of a crossbar.

which maximizes crossbar utilization while minimizing the maximum latency incurred by spikes transmitted over the interconnect. Performance is verified in terms of the spike rate and ISI at the output neuron(s) using a cycle-accurate simulator with a detailed hardware model [8]. Figure 2a shows portion of an SNN whose neurons are mapped to the 4×4 crossbar in Fig. 2b, where neuron N_1 is mapped to column 1, N_2 to column 2, and so on; λ^r and γ^r denote spike rate and average ISI, respectively, of the input spike trains.

High-voltage operations involving the crossbar cause resistance drift, electro-migration, and cell-endurance issues which reduce *inference quality* in machine-learning tasks. Inference quality for SNNs is defined in terms of ISI as follows: if $t_1, t_2, \ldots, t_k$ denote a neuron's firing times within the time interval $[0, T]$, the average ISI of this spike train is

$$\gamma = \frac{1}{(k-1)} \sum_{i=2}^{k} (t_i - t_{i-1}).$$

ISI encodes critical spatio-temporal information within SNNs; delayed, missing, or spurious spikes distort ISI and reduce inference quality. Accuracy deteriorates quickly with increasing ISI distortion, and so, we use ISI as the metric to characterize the quality of the result provided by the SNN.

III. Monitoring Approach

We develop a performance monitoring framework that uses *model-based redundancy* in which a monitor, realized in software, compares discrepancies between the behavior of neurons mapped to hardware and that predicted by a corresponding predictive model. Figure 3 shows the basic approach in the case of a single crossbar. Features related to input spike trains from pre-synaptic neurons, occurring along the crossbar's rows, are extracted by the PMU and provided to the predictive model. The PMUs use the *Address Event Representation* (*AER*) header associated with each spike packet, which identifies both the pre-synaptic and post-synaptic neurons associated with the spike, to process the spike trains of interest to the monitor. Features of interest from each spike train are the spike rate (number of spikes per unit time) and average ISI. The model, knowing the

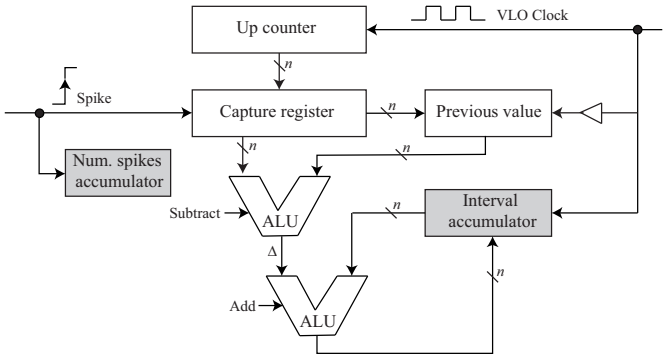


Fig. 4: Architecture of the performance monitoring unit.

mapping of synapses to that crossbar, predicts the average ISI for each of the output spike trains along crossbar columns. Simultaneously, the average ISI values from the actual spike trains occurring along the columns are extracted, again using PMUs along the columns. The predicted and actual ISI values are analyzed by a change detector to generate the error signal for that crossbar.

Figure 4 shows the architecture for the PMU which collects spike information along a single input row or output column of the crossbar. The PMU accumulates the number of observed spikes as well as the intervals between spikes within the shaded registers. To accumulate the ISI values, the circuit comprises of an up-counter controlled by a very low frequency oscillator (VLO clock). When a spike event occurs, the current counter value is latched into a capture register and the difference between the current and previous counter values is calculated. These differences are accumulated to obtain the cumulative ISI between spikes. One PMU is associated with each of the n rows in the crossbar. The AER header associated with each spike packet can be used to route the incoming spike to the correct PMU. Similarly, one PMU is associated with each of the n outgoing columns from a crossbar, and the AER information can be used to distinguish between the columns. The clock resides off-chip, requiring no changes to the neuromorphic hardware itself. Moreover, steady-state spike rates during typical SNN execution lie in the 0 to 60Hz range. These events can be accurately captured using the VLO clock operating under 10Khz while incurring very low-power consumption overhead.

For FDI, each column within the $n \times n$ crossbar is assigned a model M which accepts vector $\mathbf{I}$ as input and provides $M(\mathbf{I}) = \hat{\gamma}^c$ as output, where $\hat{\gamma}^c$ is the predicted average ISI value for the output spike train occurring along that column. The input $\mathbf{I}$ comprises of elements $(I_1, I_2, \ldots, I_n)$ where element $I_k = (\lambda_k^r, \gamma_k^r)$ consists of the spike rate λ_k^r and average ISI γ_k^r for the spike train occurring along the k^{th} row. Given input $\mathbf{I}$, predicting the continuous output value $\hat{\gamma}^c$ is a regression problem. We use *supervised approach* rather than ones derived from first principles for the following reasons: (1) significant amount of training data can be generated offline using a simulator such as CARLsim, and (2) once trained, inference is fast for real-time operation. The following models are evaluated [9]:

- *Light Gradient Boosted Machine* (*LightGBM*) is a tree-based algorithm that uses gradient-based one-side sampling technique for fast and accurate inference.
- *Category and Boosting* (*CatBoost*) uses symmetric trees to speed up inference and employs ordered boosting to avoid over-fitting to offer a better quality model.
- *ANN/CNN* is a neural network model, consisting of an input layer, at least one hidden layer (convolutional layer for CNN), and an output layer. It learns linear and non-linear relationships between the input and output.

Crossbar size $n \times n$	Mean Absolute Error (MAE)			
	ANN	CNN	LightGBM	CatBoost
32×32	0.39 ± 0.03	0.38 ± 0.05	0.37 ± 0.03	0.36 ± 0.03
64×64	0.36 ± 0.03	0.30 ± 0.05	0.33 ± 0.01	0.29 ± 0.01
128×128	0.20 ± 0.01	0.20 ± 0.04	0.18 ± 0.01	0.19 ± 0.01

Fig. 5: The MAE achieved by predictive models for different crossbar sizes. Crossbars are assumed to be fault free.

The models are trained to predict the average ISI of the spike train generated by a *single column* for all possible active crosspoints occurring along it. We define a crosspoint along a column as being *active* if its synaptic weight is programmed and thus participates in the current summation process along that column. We generate training samples for one active crosspoint occurring anywhere along the column, then concatenate to this set, samples for two active crosspoints occurring along a column, and so on up to n active crosspoints. To generate a training sample for k active crosspoints, we use k input spike trains, one for each crosspoint, generated from a Poisson distribution with firing rate of 10Hz. Values of synaptic weights at active crosspoints are chosen from a uniform distribution $[w_1, w_2]$, which are user-defined values between 0 and 1. The training dataset is normalized using min-max scaling technique. We use 80% of the dataset for training, 10% for validation, and 10% for testing. Performance is evaluated using mean absolute eror (*MAE*) which measures the average error between the ground truth and the model's prediction as $1/m \sum_{i=1}^{m} |\gamma_i^c - \hat{\gamma}_i^c|$ where m is the number of testing samples. Figure 5 lists the average MAE achieved by the models for different crossbar sizes. MAE improves as the crossbar dimension increases, for the following reason. When constructing a training sample, elements corresponding to inactive crosspoints are zeros. This results in sparsity within each sample, which is detrimental to learning. For a 32×32 crossbar, for example, the training samples have few features, and moreover, there is sparsity within these features. As the crossbar's dimension increases, the number of features increases as well. Though sparsity is present in training samples across all crossbar sizes, samples having fewer number of features are affected more by sparsity when compared to samples having more features.

A *change detector* signals a change if the run-time value γ^c deviates from its prediction $\hat{\gamma}^c$, i.e., $|\gamma^c - \hat{\gamma}^c| > th$, where th is a user-specified threshold. The basic idea being if a fault causes a crossbar's performance to deviate "appreciably" from behavior learned by the model, this change should be detected. The MAE achieved by the trained model informs the choice of th as the MAE provides the residual indicating how closely the model tracks the ground-truth values. Therefore, the threshold is chosen as $th = MAE \pm \epsilon$, where ϵ is a small margin tuned appropriately to reduce false positives.

We explain the FDI strategy using the monitoring configuration for the 4×4 crossbar previously shown in Fig. 2b. The configuration is shown in Fig. 6 in which a separate monitor is associated with each column. Each model requires only those inputs which are associated with active crosspoints; rest of the inputs are set to zero. Consider, for example, the model M_3^c uses the third column input $(I_1, I_3, I_4, 0)$ to predict $\hat{\gamma}_3^c$. The change detector ψ_3^c associated with this model examines γ_3^c and $\hat{\gamma}_3^c$, and gives a binary decision e_3^c indicating if the

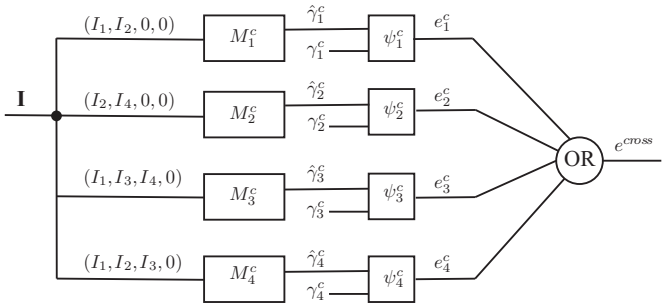


Fig. 6: Monitoring configuration for the 4×4 crossbar previously shown in Fig. 2b.

column is faulty or not. All four columns are monitored in this way and if one or more columns are determined to be faulty, the crossbar is flagged as faulty.

IV. Evaluation

Crossbar arrays of different sizes are simulated within CARLsim and fault-detection performance is evaluated on a single column within the crossbar. Since each column is monitored independently, the result generalizes readily for all n columns. Software used to generate the results is available at https://github.com/abhishekkumarm98/Online_performance_monitoring. The *functional fault model* targets two types of physical faults affecting the crossbar architecture previously shown in Fig. 1: (1) stuck-at RESET fault and (2) resistance drift within the NVM cell. A stuck-at RESET fault in which $w_i \rightarrow 0$, caused by the access transistor being stuck at 0, manifests as missing or delayed spikes in the output spike train. To inject resistance drift, the original weight is increased three-fold to $w_i \rightarrow 3 \times w_i$ based on reported findings [10] and this fault manifests as spurious spikes in the output spike train. The monitor looks for effects of these faults on the average ISI of the output spike train.

Once an SNN is mapped to hardware, some fraction of crosspoints along each column will be active. Figure 7 lists the highest detection rates achieved by the models as a function of column utilization, defined as the ratio of number of active crosspoints to n. We generate 10k samples for each crossbar wherein 5k samples contain missing and delayed spikes and the remaining contain spurious spikes. The monitor observes each spike train over non-overlapping windows of length 1000ms each, and the detection rate is the fraction of these spike trains flagged as faulty. Note that as utilization increases, the detection rate decreases; for a crossbar of size 64×64 with 10% utilization (7 active crosspoints), the detection rate for missing spikes is 89% but as utilization increases to 50% (16 active crosspoints), the detection rate decreases to 70%. This result is to be expected since as the number of crosspoints increases, the output spike train becomes less sparse and therefore, the ISI is less distorted due to a single stuck-at RESET fault. Similar behavior is observed for other crossbar sizes. For 32×32 and 64×64 crossbars at 50% utilization, an average detection rate of 71% is achieved, which is decent, but for a 128×128 crossbar, the rate is very low at 16%. Results are qualitatively similar for spurious spikes.

The computational cost for FDI is dominated by model-inference time. The system used for benchmarking is an Intel Xeon CPU operating at 2.20GHz. For the largest crossbar size (128×128) considered in our experiments, ANN incurs 433.47ms to process one column (5k samples) whereas LightGBM is significantly faster at 65.07ms. Since each column in the crossbar can be monitored independently and concurrently,

Crossbar size $n \times n$	Utilization					Model
	10%	20%	30%	40%	50%	
Missing Spikes						
32×32	0.92	0.82	0.81	0.77	0.72	CNN
64×64	0.89	0.86	0.79	0.74	0.70	LightGBM
128×128	0.71	0.55	0.37	0.24	0.16	ANN
Spurious Spikes						
32×32	0.94	0.88	0.87	0.85	0.84	CNN
64×64	0.89	0.87	0.82	0.76	0.69	LightGBM
128×128	0.83	0.67	0.52	0.38	0.30	ANN

Fig. 7: Detection rates for various crossbar utilization levels.

128 ANN models can be loaded into main memory, which requires 197.44MB of space, whereas 128 LightGBM models require 19.16MB. A *roving monitor* can extend fault coverage over multiple crossbars within the neuromorphic architecture in cost-effective manner by *time-sharing* the monitoring hardware as well as prediction models between crossbars of interest. Only a single monitor is necessary and the hardware overhead is therefore shared among several crossbars.

V. Discussion

The developed approach constitutes the first step towards dynamic redundancy wherein a combination of FDI and reconfiguration is used to tolerate failures in cost-effective fashion. When faulty SNN execution can be isolated to a crossbar, it can be locked down, and the affected neurons and synapses remapped to a spare. Execution of the SNN can be restarted on the reconfigured system. Ongoing research is addressing some limitations of this work. Average ISI, being simple to calculate in real time, was the metric used to detect changes between spike trains. Other metrics are being evaluated to improve both sensitivity and robustness of FDI.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 2008167.

References

[1] S. Song, H. Chong, A. Balaji, A. Das, J. Shackleford, and N. Kandasamy, "Dfsynthesizer: Dataflow-based synthesis of spiking neural networks to neuromorphic hardware," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 3, pp. 1–35, 2022.

[2] L. Niedermeier, K. Chen, J. Xing, A. Das, J. Kopsick, E. Scott, N. Sutton, K. Weber, N. Dutt, and J. L. Krichmar, "Carlsim 6: An open source library for large-scale, biologically detailed spiking neural network simulation," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–10.

[3] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[4] A. K. Mishra, A. Das, and N. Kandasamy, "Built-in functional testing of analog in-memory accelerators for deep neural networks," *Electronics*, vol. 11, no. 16, August 2022.

[5] G. W. Burr *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, 2017.

[6] E. Chicca *et al.*, "A vlsi recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory," *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1297–1307, 2003.

[7] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2018.

[8] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2019.

[9] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *arXiv preprint arXiv:2110.01889*, 2021.

[10] Y. Chen, L. Sun, Y. Zhou, G. M. Zewdie, V. L. Deringer, R. Mazzarello, and W. Zhang, "Chemical understanding of resistance drift suppression in ge–sn–te phase-change memory materials," *Journal of Materials Chemistry C*, vol. 8, no. 1, pp. 71–77, 2020.