On Gate Flip Errors in Computing-In-Memory

Zamshed I. Chowdhury, Hüsrev Cılasun, Salonik Resch, Masoud Zabihi, Yang Lv, Brandon Zink,
Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu
Department of Electrical and Computer Engineering, *University of Minnesota*
Minneapolis, USA 55455
{chowh005,cilas001,resc0059,zabih003,lvxxx057,zinkx030,jpwang,sachin,ukarpuzc}@umn.edu

***Abstract*—Computing-in-memory (CIM) architectures that perform logic gate operations directly within memory arrays, *in-situ*, are particularly effective in addressing memory-induced performance bottlenecks. When paired with nonvolatile memory, energy efficiency in performing bulk bitwise logic operations can reach unprecedented levels. However, unlocking this potential is not possible if functional correctness is compromised. In this paper we present a CIM-specific class of functional errors termed gate flips, where parametric variations make a logic gate behave as another. Through detailed functional and electrical characterization we demonstrate that gate flips stem from a significant subclass of write errors. Accordingly, we introduce an abstract model to enable efficient functional reliability assessment and to guide design decisions in forming universal CIM gate libraries. We also evaluate the impact on the end accuracy of computation using representative benchmarks.**

***Index Terms*—computing in memory, error model, gate flips, NVM**

I. Introduction

Conventional von Neumann machines make a distinction between compute (logic) and memory elements. Considering performance requirements of emerging data-intensive applications, the data communication overhead between compute and memory elements has long become forbidding. Blurring the physical distinction between compute and memory, Computing-in-memory (CIM) has proven itself as an attractive solution to this drawback. CIM substrates come in different flavors. Some perform computations near the memory array at the array periphery [1], [15], [24], others, directly within the arrays [6], [10], [12], [29], [31]. Further, both types of designs can use different memory technologies. Especially promising are the CIM substrates which directly use memory elements for computation by obviating any need for data to leave the memory array. Paired with nonvolatile memory (NVM) technologies, these architectures can reach unprecedented energy efficiency for bitwise logic operations. As a result, bulk bitwise operations can run in parallel across all columns or rows of the memory array, and boost the performance of numerous critical applications from machine learning [5], [16], [26], [33], genomics [3], [4], cryptography [20] or graph processing [37] domains. *Without loss of generality, in this paper we focus on this promising class of CIM substrates, which can perform universal Boolean logic gates directly in NVM [10], [12], [21], where individual memory cells can act as inputs or outputs.*

Regardless of the specific architecture or the underlying memory technology, CIM functionality heavily relies on the memory devices for both data storage and computation. Therefore, any non-ideality of memory devices such as parametric variation directly affects the reliability of CIM functions. Moreover, targeted CIM designs are compatible with traditional (CMOS) logic and incorporate conventional circuitry for control at memory cell and array granularity. Hence, they are subject to traditional sources of parameteric variation, as well. As we will demonstrate in this paper, parametric variations can trigger a CIM-specific, problematic class of functional errors where the logic function of a gate incorrectly mimics the logic function of another. We will refer to these errors as *gate flips*. By construction, if left undetected, gate flips can easily lead to silent data corruption. Unfortunately, no standard error model (typically covering memory or computation in isolation) can capture gate flips accurately, as the underlying physics depends on how the CIM substrate uses memory devices for computation. Accordingly, in this paper we

1) Introduce *gate flips*, an important class of errors affecting functional reliability of a broad class of nonvolatile CIM systems, and the *gate flip matrix*, an abstract model for efficient functional reliability assessment.
2) Investigate the underlying physical phenomena including device-level parametric variations.
3) Analyze the propagation of gate flip induced errors to the end results of computation and quantify the degradation in computational accuracy using representative benchmarks.

Chowdhury and Cılasun equally contributed to this work, which was supported in part by a Cisco Research Fellowship and NSF Grant SPX-1725420.

II. Background

A. Nonvolatile CIM Basics

Targeted CIM systems can use different NVM technologies such as MRAM [8], [17], [34] and ReRAM [12]. Due to better endurance and energy efficiency, we base our analysis on the MRAM variants [8], [17], [34]. When not used for computation, the CIM system essentially becomes an NVM array. We will refer to each such array as a tile, which supports

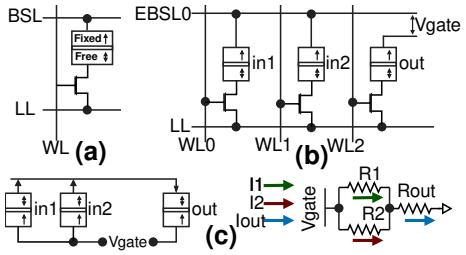

Fig. 1: (a) CIM cell architecture, (b) logic gate formation and (c) electrical equivalent circuit.

979-8-3503-4859-0/DATE24/© 2024 EDAA

TABLE I: 2-input *AND* truth table ($Output$ preset = 1).

$Input_1$	$Input_2$	$Output$	$I_{OUT} = I_1 + I_2$	
0 (R_L)	0 (R_L)	0	I_{00}	$> I_{crit}$
0 (R_L)	1 (R_H)	0	I_{01}	$> I_{crit}$
1 (R_H)	0 (R_L)	0	$I_{10} = I_{01}$	$> I_{crit}$
1 (R_H)	1 (R_H)	1	I_{11}	$< I_{crit}$

computation directly within the array, by configuring individual memory cells to act as inputs or outputs of Boolean logic gates. Each cell (Fig.1a) consists of an NVM device (spin transfer torque magnetic tunnel junction, STT-MTJ) and an access transistor that connects the cell to the control lines. Each STT-MTJ incorporates two stacked layers of ferromagnets, called fixed and free layers, separated by a thin insulating (tunneling) layer. The spin orientation of the free (fixed) layer can (not) be changed by conducting a current pulse of a particular width through the MTJ. The amplitude of the current pulse determines whether the free layer spin orientation changes, and the direction of the current pulse dictates the polarity of the spin orientation in the free layer. The relative orientation of the free and fixed layers gives rise to two distinct resistance states, low (R_L, parallel) and high (R_H, anti-parallel), which encode binary values 0 and 1, respectively.

As Fig.1a shows, one terminal of the MTJ connects to a bit select line (BSL), which spans an entire column. The other terminal connects to the logic line (LL), via a CMOS access transistor controlled by the word line (WL) spanning each row. The BSL is chunked into two groups, even and odd BSL (EBSL and OBSL), respectively. There is no difference between these groups for memory operations. For logic operations, on the other hand, inputs are expected to have a different parity (even or odd) than the output(s). A tile controller drives all signals.

Memory operations (read and write) require WL to be set, thereby selecting all cells in the corresponding row, and a (read/write-specific) voltage to be applied between LL and BSL in each column. For reads, this voltage induces a current through the selected MTJs, sensed to determine the stored logic value. This current remains lower than the critical current, $I_{crit(ical)}$, not to destroy the stored content. Write voltage, on the other hand, enforces a large enough current $> I_{crit}$ that switches the free layer orientation.

Logic gate formation along a column is demonstrated in Fig.1b: In_1 and In_2 represent the inputs; Out, the output. Before computation starts, the output is preset to a known (and gate specific) logic value. The inputs and the output are connected to BSLs of different parity (O/E BSL). A gate-specific voltage, V_{gate}, when applied between EBSL and OBSL, induces currents through the inputs, which depend on the state (resistance) of the inputs. The resulting combined current flows through the output and, depending on its magnitude, if $> I_{crit}$, may switch the output. Each Boolean gate is uniquely characterized by a preset and V_{gate} such that switching only happens according to the corresponding truth table. These CIM systems typically support various universal gates. Within a tile, while each column can perform one logic gate at a time, all (or a select subset of) columns can perform the very same logic gate (on different data, by construction) in parallel. Moreover, CIM tiles can also operate in parallel. As a representative example, Table I shows the augmented truth table of an *AND* gate. The equivalent circuit in Fig.1c illustrates how V_{gate} (=V_{AND}) induces a current through input cells (R_1 and R_2). $I_{OUT} = I_1 + I_2$, the combined current, flows through R_{OUT}, the output cell. Only for the input pattern *11*, the combined current remains $< I_{crit}$ and R_{OUT} keeps its preset value of logic 1 (R_H). Otherwise, R_{OUT} becomes logic 0 (R_L). *In a nutshell, logic operations are, essentially, write operations where the input data pattern determines whether a write at the output cell should take place or not, according to the underlying truth table*.

Application mapping spans 3 steps: (i) Application specification as a function on bitstreams; (ii) Synthesis of Boolean gates to implement this function and to allocate input, output, and scratch spaces; (iii) Binary translation to encode different V_{gate} values to implement the gate mix from (ii).

B. Errors and Parametric Variations in Nonvolatile CIM

Retention failure, failure to meet data retention requirement between successive write operations, can result in loss of data *between* memory/logic operations. Typically, a longer retention period corresponds to a higher write current through MTJ [28].

Read disturbance refers to inadvertently altering the data stored in MTJ during read operations, due to asymmetric scaling of MTJ write current and sense amplifier circuitry [25].

Write errors form a broader class. MTJ write operation is a stochastic process and depends on the duration of the write pulse, the thermal stability factor[1] and the write current [30]. The probability of write errors, write error rate (WER), corresponds to the share of failed memory (re)set operations among all attempted writes. A WER of $\leq 10^{-4}$ is achievable with a write latency of as low as 3ns [2], [22], [23], and a WER of $\leq 10^{-6}$ has been demonstrated experimentally [32]. With a write current sufficiently higher than I_{crit} and a long enough write pulse, WER can reduce to $< 10^{-8}$ [25] and become as low as 10^{-11} [9].

Stuck-at-faults emerge when the MTJ fails to change the stored value to the write value during a write operation, and can manifest in two ways: (i) the MTJ is stuck at one particular resistance level; (ii) the MTJ fails to write a particular value. (i) can be due to oxide layer breakdown or parametric variations.

Variations in MTJ I_{crit} and write latency, due to lower level parametric variations, can have grave repercussions for functional reliability, hence computational accuracy. Typically, I_{crit} follows a normal; write latency, a skewed normal distribution [19], [27].

Tunnel magneto-resistance ratio (TMR) variations stem from the variability of oxide barrier thickness in STT-MTJ [36] as well as temperature and aging [38], which in turn cause variation in the MTJ resistance. TMR is defined as $= (R_H - R_L)/R_L$.

V_{gate} variations primarily come from process variations in CMOS transistors in the drive circuitry, and directly affect the write current during logic operations. V_{gate} variations can be captured by a skewed normal distribution [35]. Variations

[1]Thermal stability factor is a physical quantity related to the period over which data can be retained.

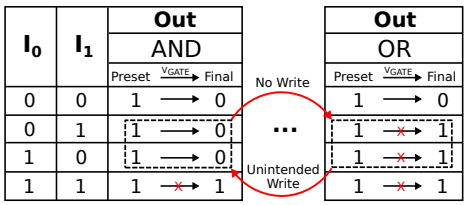

Fig. 2: Gate flip between logic *AND* and *OR*.

in transistors dominate MTJ variations for small transistors, whereas the opposite applies for larger transistors.

III. Gate Flips in Computing-In-Memory

A. Overview

Gate flip refers to functional flipping of one logic gate to another[2]. Gate flips may cover the corresponding truth table entries fully or partially. Fig.2 illustrates an example gate flip between a 2-input *AND* and a 2-input *OR* gate. The table shows the 4 input patterns (00, 01, 10, and 11) along with the *Preset* and expected (correct, *Final*) output value. The preset value 1 applies to both *OR* and *AND*. After $V_{gate} = V_{AND}$ is applied, the output of *AND* should switch ($1 \rightarrow 0$) for all input patterns except for the input pattern 11. On the other hand, for *OR*, after application of $V_{gate} = V_{OR}$, the output should switch ($1 \rightarrow 0$) for input pattern 00 only. Hence, under correct operation, the (output) switching patterns of these gates are the same for input patterns 00 and 11. The differences in the switching patterns come from input patterns 01 and 10. Consequently, if these switching patterns mimic those of the other gate, a gate flip would be the case. We will refer to these input patterns as *flip patterns*. Only the switching patterns corresponding to flip patterns can cause a gate flip between a pair of logic gates of different functionality. Therefore, we can identify flip patterns specific to pairs of logic gates.

If any of the flip patterns in *AND* results in a non-switching event instead of a switching one, i.e., a write error, the output corresponds to that of *OR*. Similarly, if the output of *OR* switches incorrectly for any of these flip patterns, the output becomes the same as that of *AND*. Although in this example we excluded corrupted presets, presets (essentially being write operations) are also susceptible to erroneous write events and can give rise to gate flips by themselves. For example, if the preset (=1) operation of *AND* fails, and the output cell's last stored value was 0, subsequent switching attempts (according to the truth table of *AND)* will fail, simply because the direction of the current through the output MTJ for a $1 \rightarrow 0$ transition cannot induce $0 \rightarrow 1$ switching due to MTJ physics. On the other hand, if the last stored value was 1, the preset error would be masked.

Gate pairs with higher number of inputs are equally susceptible to gate flips. Table II shows potential gate flips between *MAJ(ority)3* and 3-input *AND* and *OR* gates. The input patterns that have the same expected output switching pattern between {*MAJ3*,*AND*} and {*MAJ3*,*OR*} are marked by *X*. The rest of the input patterns can cause a gate flip: For *AND* (*OR*), a gate flip would be the case if $1 \rightarrow 0$ ($1 \rightarrow 1$) transitions fail. Gate flips also apply to gate pairs with different number of inputs: For example, 3-input *MAJ3* and 2-input *AND*, if one of the inputs of *MAJ3* experiences a stuck-at fault.

[2]Gate flips are fundamentally different than other gate-specific models from the literature such as "gate failures" in [14], which merely refer to incorrect gate outputs without providing any functional context.

TABLE II: Flip patterns for *MAJ3* (preset=1) to *AND* and *OR*.

I_0	I_1	I_2	MAJ3	AND	OR
0	0	0	0	X	X
0	0	1	0	X	$1 \rightarrow 1$
0	1	0	0	X	$1 \rightarrow 1$
0	1	1	1	$1 \rightarrow 0$	X
1	0	0	0	X	$1 \rightarrow 1$
1	0	1	1	$1 \rightarrow 0$	X
1	1	0	1	$1 \rightarrow 0$	X
1	1	1	1	X	X

For the sake of completeness, we cover a wide range of CIM gates, which are inherently *symmetric* (all inputs have an identical effect at the output) and *monotonic* (increasing or decreasing the number of ones or zeros at the inputs changes output behavior only once throughout the truth table rows). Gates with preset=1 include *COPY*, *AND*, *OR*, *MAJ3* (output is 1 if at least 2 inputs are 1), and *MAJ5* (output is 1 if at least 3 inputs are 1). The preset=0 duals are *NOT*, *NAND*, *NOR*, *MAJ3B* (output is 0 if at least 2 inputs are 1), and *MAJ5B* (output is 0 if at least 3 inputs are 1).

B. Gate Flips and Low-Level Errors

Gate flip is a high-level error model that captures non-ideal switching events at the output of CIM logic gates. Various low-level STT-MTJ errors can trigger the *No Write* in Fig.2, and thereby a gate flip, which causes *AND* to mimic the switching patterns of *OR*. For example, probabilistic write errors (as quantified by WER) can introduce a *No Write* event and result in an *AND* $\rightarrow$ *OR* gate flip. Also, if the write current becomes less than I_{crit}, due to a variation in either I_{crit} (triggered by parametric variations in MTJ) and/or the write current itself (triggered by fluctuations in V_{gate}), a *No Write* event can occur. Similarly, unintended writes ($1 \rightarrow 0$) for the flip patterns in Fig.2 can trigger *OR* $\rightarrow$ *AND* gate flips. Such unintended writes can be the result of variations in TMR ratio (where the actual resistance of the output cell becomes lower than expected) and/or the write current.

A stuck-at-1 fault at *AND* output can also cause an *AND* $\rightarrow$ *OR* gate flip, by keeping the output state constant at 1 during logic operations. Symmetrically, a stuck-at-0 fault at *OR* output can cause an *OR* $\rightarrow$ *AND* gate flip. However, permanent stuck-at-faults can be detected and avoided during the mapping of an application to the CIM substrate [11]. Therefore we exclude them as a potential source of gate flips. Temporary stuck-at-faults, on the other hand, manifest themselves as write errors and are hence captured by write errors in our analysis.

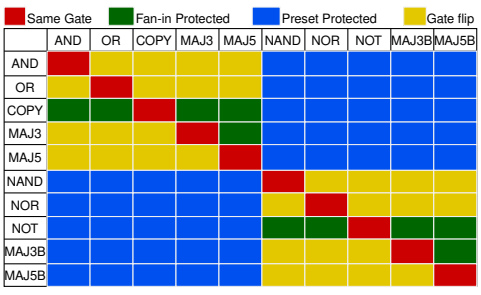

Fig. 3: Gate flip matrix. Flip direction: row $\rightarrow$ column.

C. Putting It All Together: The Gate Flip Matrix

If *S* represents the set of logic gates, then a gate in *S* can flip to any other gate in *S*, except to itself, provided that the gate pair has the following properties:

1) Both gates have the same preset: Gates that do not have the same preset, e.g., *AND* (preset = 1) and *NAND* (preset = 0), have a V_{gate} of opposite polarity, i.e., direction of current flow through the output cell is different. This difference makes a gate flip harder, if not impossible.
2) Flipping gate has an equal or higher number of inputs: For example, a 2-input *AND* can potentially flip to a 2-input *OR*, and the reverse is also true given that both logic gates have the same number of inputs. On the other hand, for a 3-input *MAJ*(ority) gate and a 2-input *AND*, the gate flip becomes uni-directional (i.e., *MAJ3* $\rightarrow$ *AND*).

Combining these rules, we introduce an abstract model, the *gate flip matrix*, for efficient functional reliability assessment and to guide design decisions in forming universal CIM gate libraries. The example gate flip matrix from Fig.3 reveals possible gate flips between each pair of gates in a representative and universal set of commonly used CIM gates, considering logic gates of different preset values such as *MAJ3* (preset = 1) and *MAJ3B* (preset = 0). We note that the majority of the gate pairs are protected from gate flips either by preset (i.e., require different preset values) or fan-in (i.e., difference in the number of inputs).

D. Impact on Functional Reliability

Functional reliability assessment should be a key step in mapping applications to CIM substrates, where abstractions like gate flip matrix can help with formal analysis and quantitative characterization. While, depending on error propagation specifics and algorithmic noise tolerance, gate-flip induced corruptions can be masked, errors induced by gate flips can also degrade the accuracy of computation.

IV. Evaluation Setup

A. Simulation Configuration

Irrespective of the underlying physical phenomena, either *failed* or *unintended* write events can cause gate flips. We capture these events with two distinct error rates, $P_{\texttt{No Write}}$ and $P_{\texttt{Err. Write}}$, respectively, which we can think of as abstractions for any lower level error and/or non-ideal behavior that can induce gate flips. Accordingly, we sweep $P_{\texttt{No Write}}$ and $P_{\texttt{Err. Write}}$ over representative ranges. Following the gate flip matrix from Fig. 3, we only consider gate flips between equal fan-in gates. We inject write errors at a random point every $\lceil \frac{1}{P} \rceil$ gate operations in a periodic fashion in the benchmark applications, where P corresponds to $P_{\texttt{No Write}}$ or $P_{\texttt{Err. Write}}$. We determine the size and number of CIM tiles depending on the problem sizes of the benchmark applications, using the following STT-MTJ parameters [21]: $R_L = 3.15$ kΩ, $R_H = 7.34$ $k\Omega$, switching time = 3 ns, $I_{crit} = 40$ μA.

B. Benchmark Applications

We consider three representative CIM benchmarks from emerging application domains: Machine learning (Support Vector Machine or SVM), graph analytics (Graph Degree Centrality or GDC), and genomics (DNA String Matching). Table III lists the benchmarks, which cover a wide range of gate flips, considering gates with different preset values. Fig.4 provides the gate mix (composition) per benchmark, excluding single-input gates. *Gmix* (*Gnand*) here denotes the mixed-gate (NAND-only) variant of the GDC benchmark. SVM, GDC, and DNA require 1024×1024 (32), 4096×1024 (96), and 128×128 (32) tiles (scratch bits), respectively. Scratch bits represent the space exclusively used for intermediate computation.

Support Vector Machine (SVM): We only cover inference with MNIST [13], grey scale 28×28 image/digit recognition dataset. The dataset has 10 classes for digits 0 through 9. Without loss of generality, we use a binarized version of the dataset. After training on the host machine, we map support vectors to separate columns of CIM tiles. The main computational kernel in SVM inference is the vector dot product operation, which boils down to a series of multiplications and additions implemented as a cascade of CIM full adders using *MAJ3* and *MAJ5* gates. More specifically, the full adder is based on a MAJ3 of all inputs, whose output is inverted twice. The inverted outputs and original inputs are then fed to a *MAJ5* gate, to generate the sum, while the *MAJ3* output corresponds to the carry output. Logic gates susceptible to gate flips in SVM inference are *AND*, *MAJ5*, and *MAJ3*. We use the percentage of misclassified test set images as the accuracy metric. The ground accuracy (excluding any errors) is 99.2% in this case, where accuracy loss corresponds to $1-$ *relative accuracy* with respect to the ground accuracy.

Graph Degree Centrality (GDC): Degree centrality in large-scale graphs captures the number of valid edges connected to a vertex. We store an adjacency matrix (covering all vertices in the input graph) in a CIM tile. Each column corresponds to a vertex and performs bit-wise addition to derive the number of valid edges connected. Hence, computation boils down to a series of CIM full adders using *MAJ3* and *MAJ5* gates. We use similar ripple-carry adders used for SVM. Therefore, in the mixed-gate variant *Gmix*, *MAJ3* and *MAJ5* are vulnerable to gate flips; as opposed to mere NAND gates in the NAND-only implementation *Gnand*. We define the accuracy loss as the average ratio of the computed degrees (under potential gate flips), to the actual node degrees.

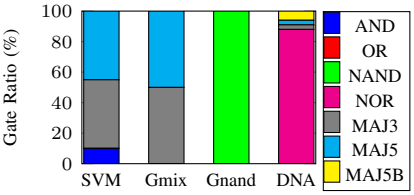

Fig. 4: Gate mix per benchmark. Gmix (Gnand) is the mixed-gate (NAND-only) variant of GDC.

Application	Dataset	Input
SVM: Support Vector Machine	MNIST	Support vectors: 51
GDC: Graph Degree Centrality	Facebook [18]	Vertices: 4039, Edges: 88234
DNA String Matching	Random	16-char references

TABLE III: Benchmarks

DNA String Matching: String matching is at the core of many critical applications, including DNA sequence alignment or database search. We map a randomly generated 16-character reference DNA database (using 2-bit encoding) to separate columns in a CIM tile of 128×128. We generate 10K 16-

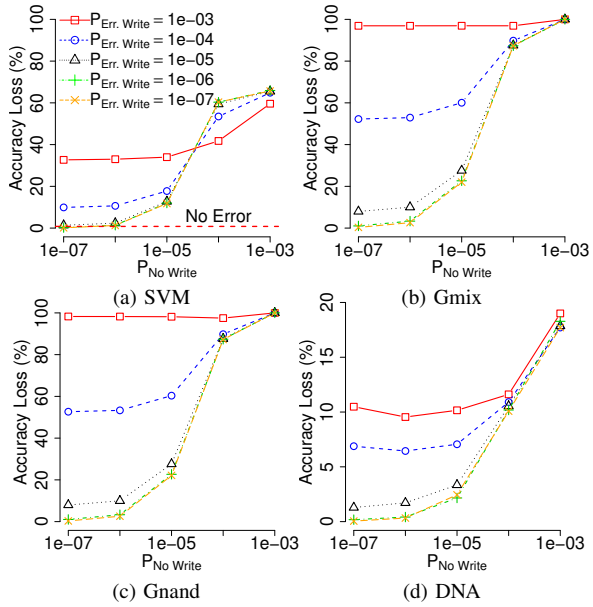

Fig. 5: Accuracy loss under gate flips.

character long random query strings to be matched against each reference, character by character. At the core of the computation, we have two *XOR* operations (implemented by *NOR* and *MAJ5B*) followed by a *NOR* to compare two DNA characters, whose outputs are added together (using *MAJ3* and *MAJ5* based full adders) to generate a similarity score. The full-adder implementation follows the SVM benchmark. The goal is to find the reference with maximum similarity to any given query string. Logic gates susceptible to gate flips in this benchmark are *NOR*, *MAJ5B*, *MAJ5* and *MAJ3*. We quantify the accuracy loss under potential gate flips by the average ratio of computed similarity score to the actual similarity score for each database entry.

V. Evaluation

A. Impact on the Accuracy of Computation

Fig.5a captures the accuracy of SVM under potential gate flips. Following intuition, accuracy degrades with higher values of $P_{\texttt{No Write}}$ and $P_{\texttt{Err. Write}}$. Under the pessimistic assumption of $P_{\texttt{No Write}} = P_{\texttt{Err. Write}} = 10^{-3}$, the accuracy loss exceeds 60% (we should note that actual write error rates are much lower, as explained in Section II). The maximum accuracy exceeds 85% for $P_{\texttt{No Write}} < 10^{-3}$, and reaches $\sim 97\%$ for $P_{\texttt{No Write}} = 10^{-6}$. This all is to be attributed to SVM's algorithmic noise tolerance. Fig.5b and Fig.5c capture the accuracy of GDC under potential gate flips for mixed-gate (*Gmix*) and NAND-only (*Gnand*) implementations. Both implementations exhibit significant accuracy loss when either of $P_{\texttt{No Write}}$ or $P_{\texttt{Err. Write}}$ exceeds 10^{-5}. Finally, Fig. 5d shows the accuracy of DNA under potential gate flips. This benchmark exhibits much higher resilience to noise than the others, as demonstrated by the mere 19% accuracy loss even when $P_{\texttt{No Write}}$=$P_{\texttt{Err. Write}}$=10^{-3}. This is mostly due to the more heterogeneous gate mix featuring *MAJ5* and *MAJ5B* gates, as larger fan-in gates tend to mask more errors.

B. Gate Flip Breakdown

We will next take a closer look at write errors. $P_{\texttt{No Write}}$ or $P_{\texttt{Err. Write}}$ just capture the probability of write errors, but there is no guarantee that an injected write error leads to a gate flip. Fig.6a depicts gate flip statistics for SVM. Y-axis shows % of total errors observed, with each stack corresponding to a specific type of error (which in general may not always represent a gate flip). The bars are grouped by $P_{\texttt{No Write}}$. We observe that, under a fixed $P_{\texttt{No Write}}$, % of *no flip* cases increases with lower $P_{\texttt{Err. Write}}$. At the same time, gate flips to *AND* (particularly from *MAJ5*) decrease with decreasing $P_{\texttt{Err. Write}}$ across the board. Since SVM inference involves a high number of additions (used in dot-product computation and multiplications), the accuracy loss peaks at $P_{\texttt{No Write}}$ and $P_{\texttt{Err. Write}}$ values, where *MAJ5* $\rightarrow$ *AND/OR* flips are dominant (Fig.5a).

As Fig.6b reveals, a similar trend applies for the mixed gate GDC implementation *Gmix*. At higher $P_{\texttt{No Write}}$ the gate flip distribution becomes more homogeneous. The homogeneous distribution is a direct result of the uniform gate mix, which features an almost equal number of *MAJ3* and *MAJ5* gates. At lower $P_{\texttt{No Write}}$, *MAJ5*$\rightarrow$*AND* flips dominate. This is when the circuit topology of the CIM full-adder implementation comes into play. As expected, the *NAND*-only implementation *Gnand* is not as sensitive to error rates (Fig.6c).

Fig.6d captures gate flip statistics for DNA. Here, unlike SVM and GDC, the gate flip distribution remains more balanced. *NOR*$\rightarrow$*NAND*, *MAJ5B*$\rightarrow$*NOR* and *MAJ5B*$\rightarrow$*NAND* flips induce errors in the first step of character comparison (*XOR*s followed by *NOR*s). At $P_{\texttt{No Write}}$ and $P_{\texttt{Err. Write}}$ values where the share of these flips is significant, the end accuracy of computation tends to degrade.

VI. Conclusion & Discussion

Computing-in-memory (CIM) is a highly effective paradigm to eliminate memory-induced performance bottlenecks. Obviating the need for data transfers, especially promising are CIM designs supporting universal Boolean logic directly in (non-volatile) memory arrays. However, they also give rise to CIM-specific errors for which no standard model exists. In this paper, we reveal one such class of errors termed gate flips, and provide a proof-of-concept characterization. We demonstrate how non-ideal effects such as parametric variations can trigger gate flips, to transform a logic gate (in memory) to a different one. We find that gate flips represent a functional abstraction for a subclass of CIM-specific write errors. We introduce a high-level model, the gate flip matrix, to enable functional reliability assessment and to guide design decisions in forming universal gate libraries for specific CIM substrates. Finally, we evaluate their impact on the end accuracy of computation using representative benchmarks from emerging application domains.

We should also note that the utility is not limited to nonvolatile CIM. Gate flips broadly apply to digital CIM architectures performing Boolean gate operations in or near the memory arrays, regardless of the underlying technology or computing mechanism at play. For example, gate flips in a CIM architecture that uses sense amplifiers (SA) at the array

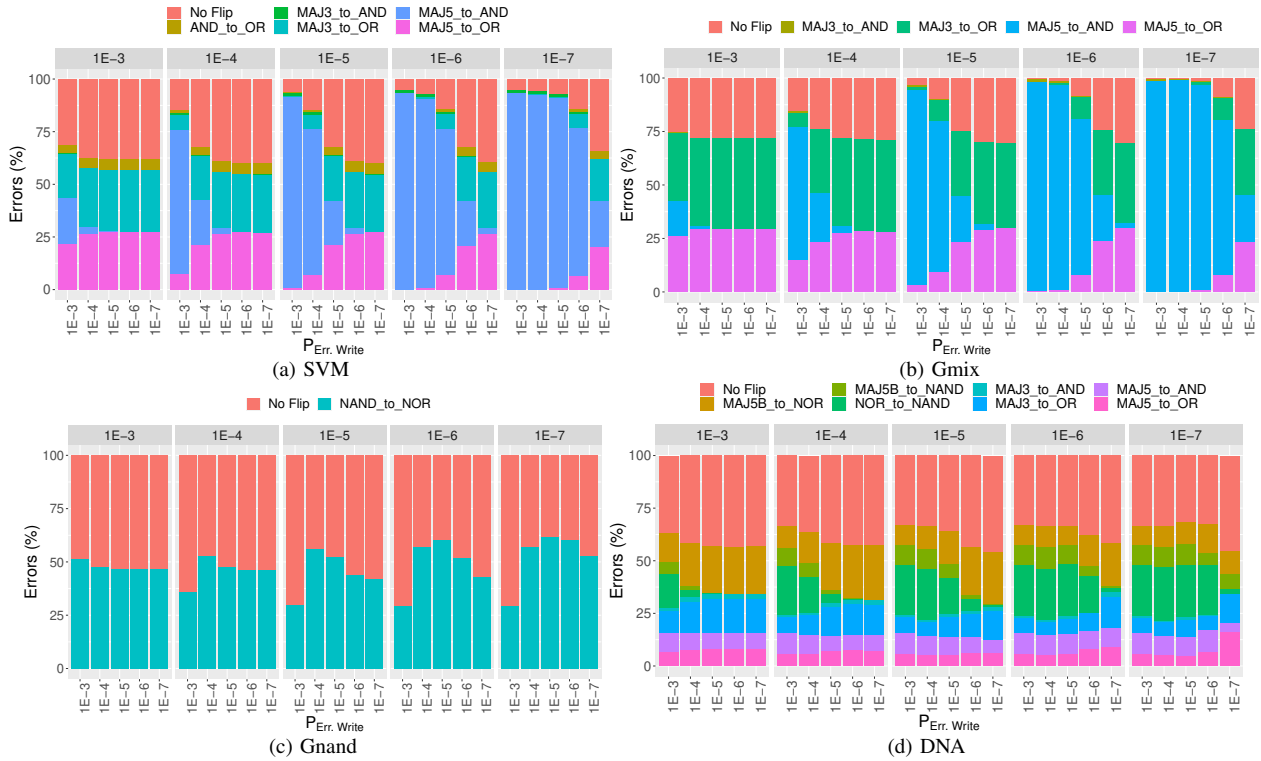

Fig. 6: Gate flip breakdown grouped by $P_{\texttt{No Write}}$ (indicated on top of each bar).

periphery to perform bit-line computing can be attributed to variations in SA threshold (which determines the output of a logic gate) [7].

REFERENCES

[1] S. Aga *et al.*, "Compute caches," in *HPCA*. IEEE, 2017.

[2] S. Aggarwal *et al.*, "Demonstration of a reliable 1 gb standalone spin-transfer torque mram for industrial applications," in *IEDM*, 2019.

[3] S. Angizi *et al.*, "Aligns: A processing-in-memory accelerator for DNA short read alignment leveraging sot-mram," in *DAC*. IEEE, 2019.

[4] Z. I. Chowdhury *et al.*, "A DNA read alignment accelerator based on computational ram," *JxCDC*, vol. 6, no. 1, 2020.

[5] H. Cılasun *et al.*, "Spiking neural networks in spintronic computational ram," *TACO*, vol. 18, no. 4, 2021.

[6] F. Gao *et al.*, "Computedram: In-memory compute using off-the-shelf drams," in *MICRO*, 2019.

[7] D. Garbin *et al.*, "Resistive memory variability: A simplified trap-assisted tunneling model," *Solid-State Electronics*, vol. 115, 2016.

[8] B. Hoffer and S. Kvatinsky, "Performing stateful logic using spin-orbit torque (sot) mram," in *NANO*. IEEE, 2022.

[9] G. Hu *et al.*, "Spin-transfer torque mram with reliable 2 ns writing for last level cache applications," in *IEDM*. IEEE, 2019.

[10] M. Imani *et al.*, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *ASP-DAC*. IEEE, 2017.

[11] G. Jung *et al.*, "Cost-and dataset-free stuck-at fault mitigation for reram-based deep learning accelerators," in *DATE*. IEEE, 2021.

[12] S. Kvatinsky *et al.*, "Magic: Memristor-aided logic," *TCAS-II*, 2014.

[13] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[14] O. Leitersdorf *et al.*, "Making memristive processing-in-memory reliable," in *ICECS*, 2021.

[15] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016.

[16] Y. Long *et al.*, "Reram-based processing-in-memory architecture for recurrent neural network acceleration," *TVLSI*, vol. 26, no. 12, 2018.

[17] J. Louis *et al.*, "Performing memristor-aided logic (magic) using stt-mram," in *ICECS*. IEEE, 2019.

[18] J. J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks." in *NeurIPS*, vol. 25, 2012.

[19] S. Motaman *et al.*, "Impact of process-variations in sttram and adaptive boosting for robustness," in *DATE*. IEEE, 2015.

[20] H. Nejatollahi *et al.*, "Cryptopim: In-memory acceleration for lattice-based cryptographic hardware," in *DAC*. IEEE, 2020.

[21] S. Resch *et al.*, "Mouse: Inference in non-volatile memory for energy harvesting applications," in *MICRO*. IEEE, 2020.

[22] D. Saida *et al.*, "Sub-3 ns pulse with sub-100μa switching of 1x–2x nm perpendicular mtj for high-performance embedded stt-mram towards sub-20nm cmos," in *ISVLSI*. IEEE, 2016.

[23] T. Saino *et al.*, "Write-error rate of nanoscale magnetic tunnel junctions in the precessional regime," *ApplPhysLetters*, vol. 115, no. 14, 2019.

[24] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *MICRO*. IEEE, 2017.

[25] J. Song *et al.*, "Impact of process variability on write error rate and read disturbance in stt-mram devices," *TMAG*, vol. 56, no. 12, 2020.

[26] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *HPCA*. IEEE, 2017.

[27] Z. Sun *et al.*, "Process variation aware data management for stt-ram cache design," in *ISLPED*, 2012.

[28] ——, "Stt-ram cache hierarchy with multiretention mtj designs," *TVLSI*, vol. 22, no. 6, 2013.

[29] J.-P. Wang and J. D. Harms, "General structure for computational random access memory (cram)," 2015, uS Patent 9,224,447.

[30] Y. Xie *et al.*, "Fokker—planck study of parameter dependence on write error slope in spin-torque switching," *T-ED*, vol. 64, no. 1, 2016.

[31] X. Xin *et al.*, "Elp2im: Efficient and low power bitwise operation processing in dram," in *HPCA*. IEEE, 2020.

[32] T. Yamamoto *et al.*, "Improvement of write error rate in voltage-driven magnetization switching," *Journal of Physics D*, 2019.

[33] X. Yang *et al.*, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *ICCAD*, 2020.

[34] M. Zabihi *et al.*, "In-memory processing on the spintronic cram: From hardware design to application mapping," *TC*, vol. 68, no. 8, 2018.

[35] Y. Zhang *et al.*, "Stt-ram cell optimization considering mtj and cmos variations," *TMAG*, vol. 47, no. 10, 2011.

[36] W. Zhao *et al.*, "Failure and reliability analysis of stt-mram," *Microelectronics Reliability*, vol. 52, no. 9-10, 2012.

[37] M. Zhou *et al.*, "Gram: graph processing in a reram-based computational memory," in *ASP-DAC*, 2019.

[38] Y. Zhou *et al.*, "A novel bist for monitoring aging/temperature by self-triggered scheme to improve the reliability of stt-mram," *Microelectron. Reliab.*, vol. 114, 2020.