# FPBA: Flexible Percentile-Based Allocation for Multiple-Bits-Per-Cell RRAM

Junfei Liu
University of Rochester, University of California San Diego
jul182@ucsd.edu

Anson Kahng
University of Rochester
anson.kahng@rochester.edu

## ABSTRACT

Advances in resistive random access memory (RRAM) technologies allow for multiple-bits-per-cell (MBPC) data storage. A central tool in MBPC data storage is a level allocation algorithm that maps resistance ranges to bit combinations. The best-performing algorithm in the literature is percentile-based allocation (PBA), which drastically improves on earlier parameterized approaches like sigma-based allocation (SBA). We demonstrate that PBA's level allocation subroutine can produce arbitrarily poor approximations of the number of levels possible at a given error threshold and propose flexible percentile-based allocation (FPBA), which is provably optimal. Additionally, we propose two heuristic interventions—finding all possible level allocations at a given error threshold and exhaustively searching over level refinements—to further reduce the bit error rate (BER) produced at the end of PBA. Our interventions result in 2.8%-32.4% lower BER and 3.1%-15.6% lower error-correcting code (ECC) storage overhead than PBA on 3- and 4-bits-per-cell (bpc) data storage schemes.

**ACM Reference Format:**
Junfei Liu and Anson Kahng. 2025. FPBA: Flexible Percentile-Based Allocation for Multiple-Bits-Per-Cell RRAM. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3658617.3697569

## 1 INTRODUCTION

Resistive random access memory (RRAM, sometimes also abbreviated ReRAM) is a non-volatile memory technology that offers significant advantages in terms of speed, power efficiency, and scalability over traditional memory architectures [17, 30]. Such improvements make RRAM suitable for a wide range of applications, including IoT devices [6, 28, 38], machine learning [4, 9, 12, 37], and monolithic 3D integration [14, 23].

One notable feature of RRAM is its ability to support multiple-bits-per-cell (MBPC) data storage schemes. Compared to traditional binary (i.e., one-bit-per-cell) storage schemes, which divide each memory cell into "low" and "high" resistance ranges, MBPC storage partitions the RRAM cell into multiple non-overlapping resistance ranges, or *levels*, which enables the storage of more than one bit. For instance, for 2 bits-per-cell (bpc) storage, the RRAM cell is partitioned into four ranges, each corresponding to a different two-digit binary number (i.e., 00, 01, 10, and 11). For RRAM in particular, previous work has demonstrated 2-4 bpc storage [13, 16, 33].

This work studies how to effectively partition MBPC RRAM cells into non-overlapping levels via a *level allocation* algorithm that maps bit combinations to resistance ranges in an RRAM cell. In a world with no errors, this task is easy. However, in practice, RRAM storage suffers from multiple analog imprecisions, including process variation [2, 15, 20, 22], relaxation effects [11, 21, 31, 34, 35], and retention errors over time [24, 25, 36], all of which have been shown to introduce significant data corruption. In order to account for unpredictable errors, level allocation algorithms typically rely on *characterization datasets* that capture real-world RRAM behavior.

**SoTA Level Allocation: SBA and PBA**. There are two main types of level allocation algorithms: parameterized distribution-based allocation, and percentile-based allocation (PBA).

Parameterized distribution-based allocation algorithms fit parameterized distributions to the characterization data and use cumulative distribution functions (CDFs) to produce low-error level allocations. The most successful of these approaches, sigma-based allocation (SBA), uses either normal or normal/log-normal distributions to model RRAM resistances [12, 16, 17, 33]. However, RRAM characterization data is often highly non-normal [32], which challenges SBA's core distributional assumption.

In contrast to parameterized allocation algorithms, PBA calculates error directly by using percentiles of the distributions in the characterization dataset, as opposed to estimating error based on (often ill-fitting) parameterizations.[1] Previous work has shown that PBA significantly outperforms SBA for both 2- and 3-bpc storage schemes on a variety of RRAM storage arrays [32], and recent work has proposed bandwidth-aware PBA (BWA-PBA) in order to make Pareto-optimal tradeoffs between write bandwidth and bit error rate (BER) [18]. We describe PBA more thoroughly in Section 2. However, PBA can return unboundedly bad level allocations. We propose a provably optimal extension of PBA as well as two heuristic improvements that markedly improve performance on real-world data.

### 1.1 Our Contributions

We propose Flexible Percentile-Based Allocation (FPBA), shown in Figure 1, a level allocation procedure that extends and improves on PBA. Our theoretical contributions are threefold:

- First, we prove the *unbounded suboptimality of PBA*: In the worst case, PBA's `LevelAlloc` (`LA`) algorithm can only find one level out of an arbitrarily high number of possible levels at any maximum acceptable error probability $\gamma$.

[1] The $k \times 100$ percentile of a distribution is the smallest value greater than a $k$ fraction of all the data. It is also equivalent to the $k^{th}$ quantile.


*ASP-DAC, January 20–23, 2025, Tokyo, Japan*

ACM ISBN 979-8-4007-0635-6/25/01
https://doi.org/10.1145/3658617.3697569

- Second, we present `FlexibleLevelAlloc` (FLA), a *provably optimal* level allocation subroutine.
- Third, we propose *two additional heuristic improvements* that can augment both LA and FLA: (1) finding all combinations of levels (`FindAllCliques`), and (2) flexible refinement of levels (FR).

In addition, we evaluate the improvement of FPBA over PBA on the publicly-available data from the paper that proposes PBA [32]:

- In our tests, FLA reduces the maximum error probability $\gamma$ by 27%–30% compared to LA for both 3 bpc and 4 bpc, but FLA alone only decreases BER for 3 bpc.
- Our two heuristic improvements, `FindAllCliques` and FR, lead to significant reductions in BER and ECC overhead over PBA's LA algorithm for both 3 bpc and 4 bpc.
- For 3 bpc, FPBA reduces the BER by 23.7%–32.4% and the ECC overhead by 11.0%–15.6%. For 4 bpc, FPBA reduces the BER by 2.8%–5.4% and the ECC overhead by 3.1%–6.3%.

## 2 BACKGROUND

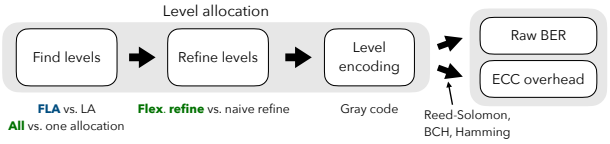


**Figure 1: (F)PBA flow. FPBA contributions bolded.**

**Percentile-Based Allocation (PBA).** We begin with a summary of PBA, whose general pipeline is depicted in Figure 1; for a more detailed description, see [32].

**PBA Level Allocation.** PBA takes two parameters as input: (1) the number of levels to fit into a resistive memory cell, $n$, and (2) a characterization dataset consisting of measurements $\langle c, t, r\rangle$, where $c$ is the resistance written to a cell and $r$ is the resistance of the cell after $t$ seconds have elapsed. PBA then uses the characterization dataset to produce an $n$-level allocation scheme consisting of non-overlapping levels $\{l_1, \ldots, l_n\}$ that minimizes the error probability, $\gamma$. Each level $l_i = \langle c_i, xl_i, xh_i\rangle$ consists of a write center $c_i$, a read interval lower bound $xl_i$, and a read interval upper bound $xh_i$, and levels $i$ and $j$ are said to be non-overlapping when the ranges $[xl_i, xh_i)$ and $[xl_j, xh_j)$ do not overlap, i.e., if and only if $\max(xl_i, xh_i) \le \min(xl_j, xh_j)$ or $\max(xl_j, xh_j) \le \min(xl_i, xh_i)$.

PBA relies on a core subroutine called LA, which finds the lowest error probability $\gamma$ that allows for $n$ non-overlapping levels. However, LA's search space is not exhaustive. For each $\gamma$, LA first modifies the distribution corresponding to each write center $c_i$ into a *candidate level* by taking all values between the $\gamma/2 \times 100$ percentile and the $(1-\gamma/2) \times 100$ percentile, which we denote $(\gamma/2, 1-\gamma/2)_i$.[2] Then, LA greedily searches for $n$ non-overlapping candidate levels. Previous work has shown that, *restricted to the candidate levels* LA *considers*, this procedure is guaranteed to return the optimal number of possible levels [32].

In order to minimize the error probability $\gamma$, PBA searches over values of $\gamma$, e.g., by doing binary search, until it finds the smallest $\gamma$ such that it can find $n$ non-overlapping candidate levels in the characterization dataset. Then, it refines these levels by "filling in the gaps" between adjacent read interval upper and lower bounds by taking the midpoint of the gaps. For instance, if two adjacent candidate levels are $\langle 10, 4, 16\rangle$ and $\langle 24, 20, 30\rangle$, then this naive refinement would result in $\langle 10, 4, 18\rangle$ and $\langle 24, 18, 30\rangle$.

**PBA Level Encoding and Evaluation.** The output of level allocation can be used to produce an $n \times n$ transition matrix $T$, where the $(i, j)^{th}$ entry denotes the frequency at which data that should be in level $i$ ends up in level $j$. In order to guarantee that an error from one level to any adjacent level only incurs a single bit flip, PBA uses Gray coding [5, 8] to assign bits to levels.

We use two evaluation metrics following [32], raw bit error rate (BER) and error-correcting code (ECC) overhead. Each can be calculated from the transition matrix $T$. The BER is defined as the ratio between the number of bit flips and the number of total bits, or BER = # bit flips/# total bits, and it can be calculated by multiplying $T$ by the Gray encoding, taking the total sum of entries in the result, and dividing by the number of bits per cell. The ECC overhead is measured based on the best overhead of a Reed-Solomon, BCH, or Hamming code with codeword size at most 12 bits that achieves a $10^{-14}$ unrecoverable bit error rate.

**It is important to distinguish between the error probability $\gamma$ minimized in LA and the overall BER.** In general, these two quantities are not the same. However, BER is both upper- and lower-bounded by a function of $\gamma$: Because each error results in at least 1 and at most $\lceil \log_2 n \rceil$ bit flips, $\gamma/\lceil \log_2 n \rceil \le \text{BER} \le \gamma$. In practice, BER may actually be *lower* than this theoretical lower bound because levels expand during refinement, further reducing the error frequency. Therefore, minimizing $\gamma$ minimizes the worst-case upper bound on BER, but a lower $\gamma$ does not necessarily imply a lower BER. In fact, we will show that a lower $\gamma$ can lead to a higher BER in our experiments for 4 bpc, i.e., $n = 16$ levels.

## 3 UNBOUNDED SUBOPTIMALITY OF PBA

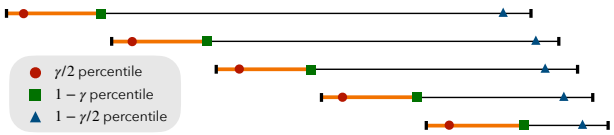


**Figure 2: Adversarial construction for PBA for $c = 5$ write centers. Red circles denote $\gamma/2\times100$ percentiles, green squares denote $(1-\gamma)\times100$ percentiles, and blue triangles denote $(1-\gamma/2)\times100$ percentiles. LA is only able to choose one level, but the optimal solution in orange chooses five.**

We begin by showing that, in the worst case, PBA's LA subroutine can perform arbitrarily poorly with respect to the optimal number of levels possible in a characterization dataset with maximum error probability $\gamma$.

Theorem 1. *For any error probability $\gamma$, target number of levels $n$, and number of distinct write centers $c \ge n$, there exists a characterization dataset with data from $c$ distinct write centers such that the maximum number of levels possible at an error rate of $\gamma$ is at least $n$, but the number of levels returned by* LA *is exactly 1.*

Proof. In the following, let $[x] := \{1, \ldots, x\}$ for $x \in \mathbb{N}$.

[2] When talking about truncated distributions in general, we omit the $i$ subscript.

Consider the following construction, depicted in Figure 2 for $c = 5$, of a characterization dataset with $c$ different write centers (distributions). For notational clarity, let us assume that the distributions are ordered in increasing order of their left-most observation.

Let $q_i^{\alpha}$ represent the $\alpha \times 100$ percentile of the $i^{th}$ distribution. In our construction, the minimum of the $(1-\gamma/2)\times 100$ percentiles of all $n$ distributions is larger than the maximum of the $\gamma/2 \times 100$ percentiles of all $n$ distributions, i.e., $\max_{i\in[c]} q_i^{\gamma/2} < \min_{i\in[c]} q_i^{1-\gamma/2}$. Additionally, for all $i \in [n-1]$, $q_i^{1-\gamma} < q_{i+1}^{0}$, i.e., the $(1-\gamma)\times 100$ percentile of the $i^{th}$ distribution is smaller than the lowest value in the $i+1^{st}$ distribution.

LA chooses exactly one distribution because all of the $(\gamma/2, 1-\gamma/2)$ ranges overlap. However, it is possible to obtain a level allocation of size $c$ with maximum error $\gamma$ by taking all $(0, 1-\gamma)$ ranges. Hence, LA obtains at most a $1/c \le 1/n$ approximation of OPT. □

# 4 FLEXIBLE PERCENTILE-BASED ALLOCATION (FPBA)

We propose **Flexible Percentile-Based Allocation (FPBA)**, an extension of PBA that fixes the unbounded suboptimality of LA and adds two heuristic improvements to LA's downstream subroutines. At the heart of FPBA is our new FlexibleLevelAlloc (FLA) algorithm, described in Section 4.1, which addresses the theoretical shortcomings of PBA's LA algorithm. We also propose two heuristic improvements, presented in Section 4.2: FindAllCliques and FlexibleRefine. Throughout, we will refer to any combination of improvements over PBA (i.e., LA) broadly as FPBA. Any omitted descriptions of subroutines are the same as in PBA.

## 4.1 Flexible Level Allocation

Our FLA algorithm is based on the following insight. In LA, the percentiles given a maximum error $\gamma$ were fixed to be $\gamma/2 \times 100\%$ and $(1-\gamma/2)\times 100\%$. However, for any $\alpha \le \gamma$, the percentiles $\alpha \times 100\%$ and $(1-\gamma+\alpha)\times 100\%$ also achieve an error probability of $\gamma$. In other words, immediately lopping off the bottom and top $\gamma/2$ of each distribution in the characterization dataset artificially constrains the possible solutions that LA can find, which limits how effectively LA can minimize $\gamma$.

FlexibleLevelAlloc (FLA), whose pseudocode is presented in Algorithm 1, is more flexible in which portions of distributions it chooses to remove. Instead of searching over only $(\gamma/2, 1-\gamma/2)$-truncated candidate levels, FLA begins with $(0, 1-\gamma)$-truncated candidate levels, i.e., the left-most $(xl, xh)$ pairs for each entry in the characterization dataset that respect the error probability $\gamma$ (see CandidateGen, lines 38–47). The FLA algorithm then proceeds much like LA and adds levels greedily according to the smallest $xh$ values remaining (see lines 23–24 and FindLeft, lines 35–36). However, as FLA fixes levels in its allocation, any candidate level with center $c_i$ that intersects with a selected level is updated to the left-most interval that contains at least a $1-\gamma$ proportion of the characterization data for center $c_i$ and does not overlap with any selected levels; if no such interval for $c_i$ exists, the candidate level is removed (see lines 18–22 and Update, lines 29–33). The FLA flow is illustrated in Figure 3.

**Algorithm 1** FLA pseudocode.

```
1:  Input:
2:    n                          ▷ Number of levels to be allocated
3:    ε                          ▷ Minimum granularity of error
4:  Define:
5:    C                          ▷ List of write centers (from dataset)
6:    t                          ▷ Relaxation time (from dataset)
7:  Output:
8:    L                          ▷ The non-overlapping level allocation
9:
10: function FlexibleLevelAlloc(n, ε)
11:     for γ ∈ [0, ε, 2ε, ..., 1] do
12:         Cands = CandidateGen(γ)
13:         Result = []
14:         valid_level, anchor = FindLeft(Cands)
15:         Result.append(valid_level)
16:         while not anchor > MAX_RESISTANCE do
17:             temp_Cands = []
18:             for (c, R, xl, xh) ∈ Cands do
19:                 if xl < anchor then
20:                     xl, xh = Update(R, anchor, xl, xh, γ)
21:                 if xl ≥ anchor then
22:                     temp_Cands.append((c, R, xl, xh))
23:             valid_level, anchor = FindLeft(temp_Cands)
24:             Result.append(valid_level)
25:             Cands = temp_Cands
26:         if length(Result) = n then
27:             return Result
28:
29: function Update(R, anchor, xl, xh, γ)
30:     left_perc = R ∈ [Min(R), anchor]      ▷ Percentage in distribution smaller than anchor
31:     if left_perc > γ then
32:         return xl, ∞
33:     return anchor, Percentile(R, 1 − (γ − left_perc))
34:
35: function FindLeft(Cands)
36:     return MIN(Cands, key = lambda x : x[xh])   ▷ Find and return candidate with smallest xh value
37:
38: function CandidateGen(γ)
39:     Cands = []                 ▷ Candidate levels for all write ranges
40:     for c ∈ C do
41:         R = data_prov(c, t)    ▷ Read characterization dataset
42:         R = sort(R)
43:         xl = Percentile(R, 0)                ▷ Lower bound
44:         xh = Percentile(R, 1 − γ)            ▷ Upper bound
45:         assert Probability(R ∈ [xl, xh] ≤ 1 − γ)
46:         Cands.Append((c, R, xl, xh))
47:     return Cands
48:
49: function Percentile(R, perc)
50:     index = perc · size(R)
51:     return R_sorted[index]
```

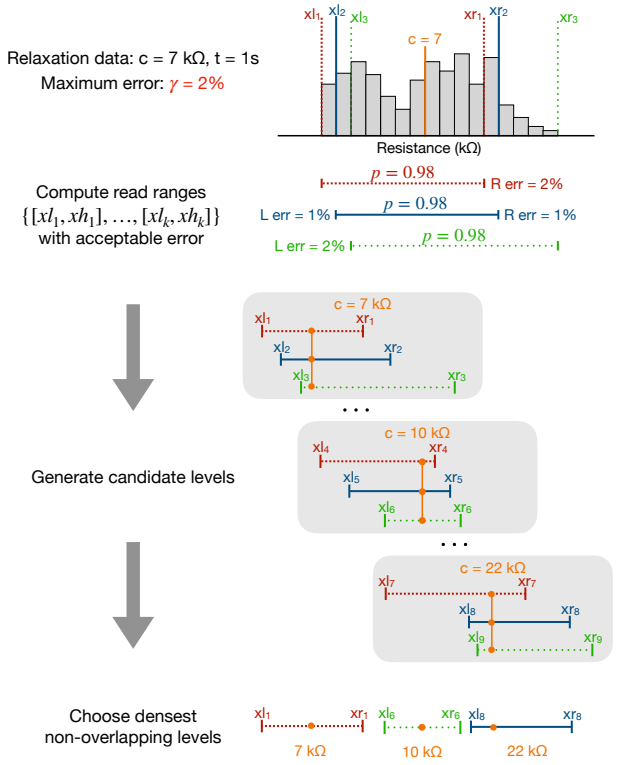


**Figure 3: Flexible level allocation (FLA). For a maximum error probability of $\gamma = 2\%$, the pairs of dashed red lines, solid blue lines, and dotted green lines each correspond to one candidate level based on the relaxation data at the top.**

*4.1.1 Optimality of* FLA. We now prove that FLA is optimal for the problem of minimizing the error probability $\gamma$.

Theorem 2. *For a given error $\gamma$ and any input characterization dataset,* FLA *returns an allocation with the optimal number of levels.*

The proof follows that in [32] (i.e., "Greedy stays ahead"). In the interest of space, we do not fully recreate the proof here.

Proof sketch. Let $\mathcal{L}$ be a list of *all possible candidate levels*, i.e., all $\langle c_i, xl_i, xh_i\rangle$ such that the range $[xl_i, xh_i)$ contains at least $\gamma$ of the characterization data from center $c_i$, sorted by increasing upper read resistance. Note that, unlike in LA, there may be multiple candidate levels per center. Let the indices of levels returned by FLA be $\{x_1, \ldots, x_k\}$, and assume toward a contradiction that there is a better solution that consists of the indices $\{y_1, \ldots, y_\ell\}$ for $\ell > k$. For each level at index $z_i$, let $lr(z_i)$ and $ur(z_i)$ denote the lower and upper read resistance, respectively.

We will show that this better solution cannot exist. First, we prove by induction *Claim 1*: for all $i \le k$, we have $ur(x_i) \le ur(y_i)$.

Now, if $\ell > k$, then for the $(k+1)^{st}$ level in our better solution, we must have $lr(y_{k+1}) \ge ur(y_k)$ because levels cannot overlap. Because $ur(x_k) \le ur(y_k)$ by Claim 1 (proved via induction), this means that the level at $y_{k+1}$ is compatible with all levels in the solution returned by FLA. Therefore, FLA would have added this level to its solution, which is a contradiction. We may conclude that FLA returns an optimal solution. □

## 4.2 Heuristic Improvements

In addition to FLA, we present two additional heuristic improvements to PBA, presented in green in Figure 1: (1) finding all combinations of levels, and (2) flexibly refining level allocations. As we show in our experiments, these interventions have the potential to significantly reduce BER and ECC overhead, with intervention (1) being particularly effective.

*4.2.1 Find All Levels (*`FindAllCliques`*, or* AC*).* Both LA and FLA find a single level allocation—in fact, the lexicographically first allocation—satisfying the minimum $\gamma$ requirement. However, there are often multiple different level allocations that achieve a given $\gamma$ error target, and choosing the lexicographically earliest allocation is not guaranteed to result in (close to) the best performance.

Given an error $\gamma$, we propose a graph-based approach applicable to both LA and FLA to find all admissible level allocations. First, define the set of candidate levels according to $\gamma$. For LA+AC, there will be as many candidates as the number of write centers, as described by CandidateGen($\gamma$) in Algorithm 1; for FLA+AC, we generate every possible read interval that contains at least $1-\gamma$ of the write distribution for each write center as candidate levels. Then, we may construct the *level graph* $G = (V, E)$: each vertex corresponds to one candidate level, and two vertices are connected iff their respective candidate levels do not overlap. Now, the "find all $n$-level allocations" problem becomes exactly "find all cliques of size $n$ in $G$," which we denote `FindAllCliques`$(G, n)$.

*4.2.2 Level Refinement (*`FlexibleRefine`*).* We may also explore all possible ways of refining the $n$ candidate levels returned by either LA or FLA. Recall that PBA uses the following naive refinement method: given any two adjacent levels $l_1 = \langle c_1, xl_1, xh_1\rangle$ and $l_2 = \langle c_2, xl_2, xh_2\rangle$, if $xh_1 < xl_2$, then both $xh_1$ and $xl_2$ "expand" to their midpoint, $\frac{xh_1+xl_2}{2}$, resulting in levels $\langle c_1, xl_1, \frac{xh_1+xl_2}{2}\rangle$ and $\langle c_2, \frac{xh_1+xl_2}{2}, xh_2\rangle$. Additionally, the first and last levels adjust their $xl$ and $xh$, respectively, to the minimum and maximum values possible. While this naive refinement method works well in practice, there is opportunity for improvement by exhaustively exploring all refinement options, which we term `FlexibleRefine` (FR).

# 5 EXPERIMENTS

We now compare the performance of FPBA and PBA in simulation. In this study, we leverage data from two Ember chips, as collected by Wei et al. [32], to evaluate our proposed methods.[3] The Ember chips [27], fabricated using a 40 nm CMOS process with RRAM bit cells, feature an on-board write-verify algorithm and a 6-bit, 64-level ADC. These chips provide data by exercising 16,384 cells at 64 write centers with a 2-level tolerance, measured at 0 and 1 second. The relaxation time is 1 second and the writing algorithm is on-chip ISPP. We use the established methodology for calculating the BER and the ECC overhead in [32], which is also described in Section 2. This involves generating level allocations for the characterized RRAM storage array and employing a minimum error granularity $\epsilon$ of $10^{-6}$. The implementation searches over 64 ADC resistance values with a write tolerance of 2 ADC resistance values.

Retention and relaxation are critical challenges in RRAM. For our study, we use data collected 1 second after programming to

[3] Our code is available at https://github.com/JeffLiu114514/FlexiblePBA.

| Chip | bpc | Max Error Prob $\gamma$ | | | | Bit Error Rate (BER) | | | | ECC Overhead | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LA | FLA | $\Delta\gamma$ | % $\Delta\gamma$ | LA | FLA | $\Delta$BER | % $\Delta$BER | LA | FLA | $\Delta$ECC | % $\Delta$ECC |
| Ember1 | 3 | 2.2% | **1.6%** | -0.68% | -30% | 0.38% | **0.35%** | -0.03% | -7.8% | **9.1%** | **9.1%** | 0% | 0% |
| Ember2 | 3 | 3.0% | **1.9%** | -1.2% | -39% | 0.37% | **0.35%** | -0.015% | -4.2% | **9%** | **9%** | 0% | 0% |
| Ember1 | 4 | 26% | **19%** | -7.0% | -27% | **3.6%** | 3.7% | 0.015% | 0.4% | **32%** | **32%** | 0% | 0% |
| Ember2 | 4 | 30% | **21%** | -9.2% | -30% | **3.7%** | 4.0% | 0.36% | 9.9% | **32%** | **32%** | 0% | 0% |

**Table 1: Comparison between PBA's LA and FPBA's FLA level allocation algorithms for 3 and 4 bpc. Lower $\gamma$, BER, and ECC overhead are better. % $\Delta X$ denotes the relative percentage change in metric $X$ from the LA baseline.**

| Chip | bpc | Bit Error Rate (BER) | | | | | | ECC Overhead | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LA | LA+AC | % $\Delta$BER | FLA | FLA+AC | % $\Delta$BER | LA | LA+AC | % $\Delta$ECC | FLA | FLA+AC | % $\Delta$ECC |
| Ember1 | 3 | 0.38% | **0.29%** | -24% | 0.35% | **0.29%** | -18% | 9.1% | **8.1%** | -11% | 9.1% | **8.1%** | -11% |
| Ember2 | 3 | 0.37% | 0.26% | -29% | 0.35% | **0.25%** | -30% | 9.0% | 7.7% | -15% | 9.0% | **7.6%** | -16% |
| Ember1 | 4 | 3.6% | 3.5% | -1.2% | 3.7% | - | - | 32% | 31% | -1.4% | 32% | - | - |
| Ember2 | 4 | 3.7% | 3.5% | -3.1% | 4.0% | - | - | 32% | 31% | -2.1% | 32% | - | - |

**Table 2: Ablation study for FindAllCliques allocation (AC). Lower BER and ECC overhead are better. AC = FindAllCliques and % $\Delta X$ denotes the relative percentage change in metric $X$ from the relevant baseline (e.g., comparing FLA+AC vs. FLA).**

align with the methodology employed in [27], which allows for consistent comparison against PBA. Previous work has found that significant resistance drift occurs within a short time window (< 1 second) following programming, after which the drift slows considerably [29]. Measuring retention over long periods (e.g., ∼10 years) is impractical, making the 1-second data a reasonable compromise.

We primarily report a comparison between PBA and our methods for 3 bpc (8 levels) and 4 bpc (16 levels) in our experimental results. We do not include 2 bpc results since both PBA and our improvements already achieve 0% BER and ECC overhead. We use PBA as our baseline because it has been shown to strongly outperform prior state-of-the-art algorithms such as SBA [17, 32].

*5.0.1* `FindAllCliques` *Subroutine.* In our experiments, we use the Python package NetworkX to create and analyze the level graph proposed in Section 4.2.1. Because any clique of size $n$ is contained in some maximal clique of size $\geq n$, it suffices to enumerate all subsets of size exactly $n$ of all maximal cliques of size at least $n$. However, the worst-case time complexity for enumerating all maximal cliques in a graph $G = (V, E)$ is $O(3^{|V|/3})$, with $O(|V|)$ space complexity, where $|V|$ is the number of vertices in $G$ [1, 3, 26]. Therefore, running `FindAllCliques` in conjunction with `FLA` is exceptionally time-intensive due to the large number of vertices (i.e., candidate levels) considered by `FLA`.

## 5.1 Results

We compare the performance of PBA's `LA` algorithm against FPBA's `FLA` algorithm in Table 1. In all cases, as expected, `FLA` finds lower values of maximum error probability $\gamma$ than `LA`. However, with respect to BER, `FLA` outperforms `LA` for 3 bpc on both Ember chips but does slightly worse for 4 bpc. There is no change in ECC overhead because the BER changes in both directions are relatively small. Overall, this demonstrates that reducing $\gamma$ does not always reduce BER, so we turn to our heuristic improvements to see if they can reduce $\gamma$ and BER simultaneously.

Adding in the `FindAllCliques` subroutine significantly improves the performance of FPBA over PBA. In Table 2, we compare the performance of PBA's `LA` algorithm against `LA` with `FindAllCliques` (`LA+AC`) as well as `FLA` with `FindAllCliques` (`FLA+AC`), and see that adding `FindAllCliques` drastically outperforms the base versions of both `LA` and `FLA`. Our method reduces BER by 23.9%-32.4% for 3 bpc, and 2.7%-5.4% for 4 bpc. We also observe reductions in ECC overhead of 11.0%-15.6% for 3 bpc, and 3.1% for 4 bpc.

We do not present results for `FLA+AC` for 4 bpc because running `FindAllCliques` on the large graphs generated after `FLA` is intractable. However, augmenting `LA` with `FindAllCliques` already yields significant improvements over PBA.

| Method | Ember1 (3 bpc) | Ember2 (3 bpc) | Ember1 (4 bpc) | Ember2 (4 bpc) |
|---|---|---|---|---|
| LA | 0.38%, 9.1% | 0.37%, 9.0% | 3.6%, 32% | 3.7%, 32% |
| LA+FR | 0.38%, 9.1% | 0.34%, 9.0% | **3.5%**, 32% | 3.7%, 32% |
| FLA | 0.35%, 9.1% | 0.35%, 9.0% | 3.7%, 32% | 4.0%, 32% |
| FLA+FR | 0.33%, 9.1% | 0.34%, 9.0% | 3.7%, 32% | 3.9%, 32% |
| LA+AC | **0.29%**, **8.1%** | 0.26%, 7.7% | 3.6%, 31% | **3.5%**, **31%** |
| **LA+AC+FR** | **0.29%**, **8.1%** | **0.25%**, **7.6%** | **3.5%**, **30%** | **3.5%**, **31%** |
| **FLA+AC** | **0.29%**, **8.1%** | **0.25%**, 7.7% | – | – |
| **FLA+AC+FR** | **0.29%**, **8.1%** | **0.25%**, 7.7% | – | – |

**Table 3: Ablation study for FlexibleRefine (FR) on BER and ECC overhead. Format in each cell is BER, ECC overhead. For both BER and ECC overhead, lower is better. AC = FindAllCliques and FR = FlexibleRefine.**

We next add our second heuristic improvement, flexible refinement (`FR`), in experiments summarized in Table 3. Overall, we note that although `FR` demonstrates modest reductions in BER (8.1%-13.2%) for 3 bpc, its improvements are much smaller for 4 bpc, and it reduces ECC overhead by only at most 1.2%–3.2% in any of our tests (from `LA+AC` to `LA+AC+FR` for both 3 and 4 bpc). This is likely because refining the boundaries between levels can only marginally impact overall performance; especially for 4 bpc, the level allocation is already so dense, there isn't much room to even change boundaries between levels.

Our overall best results are bolded in green in Table 3. In particular, `LA+AC+FR` and `FLA+AC+FR` consistently offer improvements over PBA (i.e., `LA`). For 3 bpc, both setups reduce the BER by 23.7% on Ember1 and 32.4% on Ember2, and the ECC overhead by 11.0% on Ember 1 and 14.4%–15.6% on Ember2 (`LA+AC+FR` performs slightly better). For 4 bpc, `LA+AC+FR` reduces the BER by 2.8% on Ember1

and 5.4% on Ember2, and the ECC overhead by 6.3% on Ember1 and 3.1% on Ember2. However, neither is perfect: LA+AC+FR suffers from the same theoretical shortcomings as PBA, and FLA+AC+FR is expensive to run for large $n$.

| Size | Ember1 Bit Error Rate | | | |
|---|---|---|---|---|
| | LA | FLA | LA+AC | FLA+AC |
| 25% | 0.58 ± 0.16 | 0.58 ± 0.16 | **0.57** ±0.17 | **0.57** ±0.15 |
| 50% | 0.52 ± 0.10 | 0.47 ± 0.07 | 0.41 ± 0.05 | **0.39** ±0.05 |
| 75% | 0.42 ± 0.07 | 0.41 ± 0.04 | 0.34 ± 0.04 | **0.32** ±0.04 |
| 90% | 0.38 ± 0.04 | 0.37 ± 0.04 | **0.29** ±0.01 | **0.29**±0.01 |
| 100% | 0.38 | 0.35 | **0.29** | **0.29** |
| Size | Ember2 Bit Error Rate | | | |
| | LA | FLA | LA+AC | FLA+AC |
| 25% | 0.5 ± 0.12 | 0.5 ± 0.12 | **0.47** ±0.11 | 0.55 ± 0.12 |
| 50% | 0.58 ± 0.09 | 0.53 ± 0.08 | **0.36** ±0.09 | 0.41 ± 0.09 |
| 75% | 0.43 ± 0.06 | 0.46 ± 0.06 | 0.29 ± 0.04 | **0.27** ±0.03 |
| 90% | 0.46 ± 0.10 | 0.38 ± 0.06 | **0.26** ±0.03 | 0.27 ± 0.04 |
| 100% | 0.37 | 0.35 | 0.26 | **0.25** |

**Table 4: Effect of dataset size on Bit Error Rate (BER) in level allocation (3 bpc), 10 samples each. All BER values in %.**

Inspired by similar experiments in [32], we run experiments to examine the effect of sample size (Table 4) and multi-chip data (Table 5) on BER and ECC overhead.

For sample size, we repeatedly sample a given percentage from the resistance distributions of each write center 10 times and present the mean and standard deviation in Table 4. We observe that FLA slightly improves on LA's performance, but LA+AC and FLA+AC strongly outperform both LA and FLA for all subset sizes, suggesting that FPBA as a general framework—and the specific combinations of LA+AC and FLA+AC in particular—is more robust to missing data than PBA.

| Mix | Ember1 Bit Error Rate | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | LA | LA+FR | FLA | FLA+FR | LA+AC | LA+AC+FR | FLA+AC | FLA+AC+FR |
| 100/0 | 0.38 | 0.38 | 0.35 | 0.33 | **0.29** | **0.29** | **0.29** | **0.29** |
| 50/50 | 0.46 | 0.39 | 0.56 | 0.51 | 0.46 | **0.38** | 0.44 | 0.41 |
| 10/90 | 0.65 | **0.63** | 0.78 | 0.71 | 0.69 | 0.67 | 0.67 | 0.67 |
| 0/100 | **0.64** | **0.64** | 0.71 | 0.71 | 0.69 | 0.67 | 0.67 | 0.67 |
| Mix | Ember2 Bit Error Rate | | | | | | | |
| | LA | LA+FR | FLA | FLA+FR | LA+AC | LA+AC+FR | FLA+AC | FLA+AC+FR |
| 100/0 | 0.37 | 0.34 | 0.35 | 0.34 | 0.26 | **0.25** | **0.25** | **0.25** |
| 50/50 | 0.49 | 0.51 | 0.52 | 0.54 | 0.47 | 0.47 | 0.35 | **0.32** |
| 10/90 | 0.64 | 0.66 | 0.81 | 0.81 | **0.55** | 0.63 | 0.77 | 0.63 |
| 0/100 | 0.72 | 0.72 | 0.8 | 0.8 | **0.67** | **0.67** | **0.67** | **0.67** |

**Table 5: Interchip level allocation BER comparison (3 bpc). A mix of $x/y$ = $x$% target chip's dataset plus $y$% from another chip. All BER values in %.**

With respect to multi-chip data, we create mixtures of data from Ember1 and Ember2, where the notation $x/y$ means that $x$% of the data came from the target chip's characterization dataset and $y\% = (100 - x)\%$ of the data came from the other chip's characterization dataset. We then evaluate the BER and ECC overhead for various forms of FPBA using these mixtures of data for both Ember1 and Ember2 as target chips. Overall, we see that various forms of FPBA outperform PBA over all mixtures, but the performance of more comprehensive flows (i.e., LA+AC+FR and FLA+AC+FR) degrades as the mixtures become less accurate. This suggests that our more complicated methods overfit to specific chip data.

# 6 DISCUSSION

While our FLA procedure provably minimizes the error probability $\gamma$ achievable with $n$ levels, FPBA is not a formal end-to-end procedure for minimizing BER or ECC overhead. Additionally, just like PBA [32], our method is explicitly data-dependent, which comes with both positives (we can produce effective allocations based on historical data, and we may continue updating characterization datasets in order to update allocations over time) and negatives (it may not be feasible to collect characterization datasets for every fabricated chip). Furthermore, our full FLA+AC+FR procedure is computationally expensive due to the complexity of `FindAllCliques` on large graphs. However, in practice it is worth running an expensive procedure once in order to fabricate efficient RRAM. Alternatively, it may be worth exploring approximate versions of `FindAllCliques` that sample from the set of maximal cliques.

We also considered two additional heuristic improvements: (3) relaxing $\gamma$ and (4) going beyond Gray coding. The relationship between $\gamma$ and BER is not clearly monotonic, and our experiments demonstrate that a lower $\gamma$ may lead to a higher BER and ECC overhead, although $\gamma$ is a useful upper bound on BER. For both LA and FLA, after determining the smallest achievable value of $\gamma$, we may run the relevant allocation procedure on relaxed values of $\gamma$; e.g., if we relax up to 3x, we would consider $\gamma' = \gamma \cdot \alpha$ for $\alpha \in \{1.1, 1.2, \ldots, 3\}$. However, our initial experiments with relaxing $\gamma$ even up to 5x showed negligible improvement. Also, while we can construct (very adversarial) level distributions that make Gray coding arbitrarily suboptimal by forcing all errors to "jump" multiple levels, we found by brute force that Gray coding was the optimal encoding for the 2- and 3-bpc settings in our experiments.

# 7 RELATED WORK

Our work is mostly closely related to, and in fact directly extends, the results in [32]. We directly improve on the LA subroutine from PBA, and we keep the rest of the evaluation pipeline downstream of level allocation the same.

Some of our extensions of PBA are independently similar to those in [18]. In particular, the way we construct our clique graph in our `FindAllCliques` subroutine is similar to their strategy of creating a directed graph among candidate levels. However, they do not consider all candidate levels with error at most $\gamma$, and finding all paths from a source to a sink in the directed graph is not as flexible as our `FindAllCliques` subroutine. Additionally, although their methods of calculating optimal read boundaries between adjacent levels are similar to our exhaustive `FlexibleRefine` subroutine, their overall goal of computing Pareto-optimal allocations for write bandwidth and BER also differs from our goal of minimizing BER and ECC overhead.

More broadly, our work connects to a broad literature on MBPC storage in RRAM and phase-change memories (PCM) that considers data distributions when designing storage algorithms [7, 10, 19].

# ACKNOWLEDGMENTS

We would like to thank Anjiang Wei and Akash Levy for their helpful correspondence about PBA, as well as Andrew Kahng for bringing this problem to our attention.

# REFERENCES


[1] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

[2] E Brum, Moritz Fieback, Thiago Santos Copetti, H Jiayi, Said Hamdioui, Fabian Vargas, and LM Bolzani Poehls. 2021. Evaluating the impact of process variation on RRAMs. In *2021 IEEE 22nd Latin American Test Symposium (LATS)*. IEEE, 1–6.

[3] Frédéric Cazals and Chinmay Karande. 2008. A note on the problem of reporting maximal cliques. *Theoretical computer science* 407, 1-3 (2008), 564–568.

[4] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 27–39.

[5] Robert W Doran. 2007. *The Gray code*. Technical Report. University of Auckland, Department of Computer Science.

[6] Massimo Giordano, Kartik Prabhu, Kalhan Koul, Robert M Radway, Albert Gural, Rohan Doshi, Zainab F Khan, John W Kustin, Timothy Liu, Gregorio B Lopes, et al. 2021. CHIMERA: A 0.92 TOPS, 2.2 TOPS/W edge AI accelerator with 2 MByte on-chip foundry resistive RAM for efficient training and inference. In *2021 symposium on VLSI circuits*. IEEE, 1–2.

[7] Artem Glukhov, Valerio Milo, Andrea Baroni, Nicola Lepri, Cristian Zambelli, Piero Olivo, Eduardo Pérez, Christian Wenger, and Daniele Ielmini. 2022. Statistical model of program/verify algorithms in resistive-switching memories for in-memory neural network accelerators. In *2022 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 3C–3.

[8] Frank Gray. U.S. Patent #2,632,058, March 17, 1953. Pulse Code Communication.

[9] Alessandro Grossi, Elisa Vianello, Mohamed M Sabry, Marios Barlas, Laurent Grenouillet, Jean Coignus, Edith Beigne, Tony Wu, Binh Q Le, Mary K Wootters, et al. 2019. Resistive RAM endurance: Array-level characterization and correction techniques targeting deep learning applications. *IEEE Transactions on Electron Devices* 66, 3 (2019), 1281–1288.

[10] Qing Guo, Karin Strauss, Luis Ceze, and Henrique S Malvar. 2016. High-density image storage using approximate memory cells. *ACM SIGPLAN Notices* 51, 4 (2016), 413–426.

[11] Wangxin He, Wonbo Shim, Shihui Yin, Xiaoyu Sun, Deliang Fan, Shimeng Yu, and Jae-sun Seo. 2021. Characterization and mitigation of relaxation effects on multi-level RRAM based in-memory computing. In *2021 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 1–7.

[12] ER Hsieh, M Giordano, B Hodson, A Levy, SK Osekowsky, RM Radway, YC Shih, W Wan, TF Wu, X Zheng, et al. 2019. High-density multiple bits-per-cell 1T4R RRAM array with gradual SET/RESET and its effectiveness for deep learning. In *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 35–6.

[13] ER Hsieh, X Zheng, BQ Le, YC Shih, RM Radway, M Nelson, S Mitra, and S Wong. 2021. Four-bits-per-memory one-transistor-and-eight-resistive-random-access-memory (1T8R) array. *IEEE Electron Device Letters* 42, 3 (2021), 335–338.

[14] Ping-Yi Hsieh, Yi-Jui Chang, Pin-Jun Chen, Chun-Liang Chen, Chih-Chao Yang, Po-Tsang Huang, Yi-Jing Chen, Chih-Ming Shen, Yu-Wei Liu, Chien-Chi Huang, et al. 2019. Monolithic 3D BEOL FinFET switch arrays using location-controlled-grain technique in voltage regulator with better FOM than 2D regulators. In *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 3–1.

[15] A Kalantarian, G Bersuker, DC Gilmer, D Veksler, B Butcher, Andrea Padovani, Onofrio Pirrotta, Luca Larcher, R Geer, Y Nishi, et al. 2012. Controlling uniformity of RRAM characteristics through the forming process. In *2012 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 6C–4.

[16] Binh Q Le, Alessandro Grossi, Elisa Vianello, Tony Wu, Giusy Lama, Edith Beigne, H-S Philip Wong, and Subhasish Mitra. 2018. Resistive RAM with multiple bits per cell: Array-level demonstration of 3 bits per cell. *IEEE Transactions on Electron Devices* 66, 1 (2018), 641–646.

[17] Binh Q Le, Akash Levy, Tony F Wu, Robert M Radway, E Ray Hsieh, Xin Zheng, Mark Nelson, Priyanka Raina, H-S Philip Wong, Simon Wong, et al. 2021. RADAR: A fast and energy-efficient programming technique for multiple bits-per-cell RRAM arrays. *IEEE Transactions on Electron Devices* 68, 9 (2021), 4397–4403.

[18] Akash Levy, Luke R Upton, Michael D Scott, Dennis Rich, Win-San Khwa, Yu-Der Chih, Meng-Fan Chang, Subhasish Mitra, Boris Murmann, and Priyanka Raina. 2024. EMBER: Efficient Multiple-Bits-Per-Cell Embedded RRAM Macro for High-Density Digital Storage. *IEEE Journal of Solid-State Circuits* (2024).

[19] Boxun Li, Peng Gu, Yi Shan, Yu Wang, Yiran Chen, and Huazhong Yang. 2015. RRAM-based analog approximate computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 12 (2015), 1905–1917.

[20] Haitong Li, Zizhen Jiang, Peng Huang, Yi Wu, H-Y Chen, Bin Gao, XY Liu, JF Kang, and H-SP Wong. 2015. Variation-aware, reliability-emphasized design and optimization of RRAM using SPICE model. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1425–1430.

[21] Yuyi Liu, Bin Gao, Feng Xu, Wenqiang Zhang, Yue Xi, Jianshi Tang, and He Qian. 2021. A compact model for relaxation effect in analog RRAM for computation-in-memory system design and benchmark. In *2021 5th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. IEEE, 1–3.

[22] Peyman Pouyan, Esteve Amat, Said Hamdioui, and Antonio Rubio. 2016. RRAM variability and its mitigation schemes. In *2016 26th international workshop on power and timing modeling, optimization and simulation (PATMOS)*. IEEE, 141–146.

[23] Max M Shulaker, Tony F Wu, Mohamed M Sabry, Hai Wei, H-S Philip Wong, and Subhasish Mitra. 2015. Monolithic 3D integration: A path from concept to reality. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1197–1202.

[24] Wendong Song, Hock Koon Lee, Weijie Wang, Minghua Li, Zhixian Chen, Jen-Chieh Liu, I-Ting Wang, Victor Yi-Qian Zhuo, and Yao Zhu. 2020. Investigation of Retention Failure Behavior in Analog RRAM Devices. In *2020 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*. IEEE, 1–4.

[25] Zainab Swaidan, Rouwaida Kanj, Johnny El Hajj, Edward Saad, and Fadi Kurdahi. 2019. Rram endurance and retention: Challenges, opportunities and implications on reliable design. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 402–405.

[26] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science* 363, 1 (2006), 28–42.

[27] Luke R. Upton, Akash Levy, Michael D. Scott, Dennis Rich, Win-San Khwa, Yu-Der Chih, Meng-Fan Chang, Subhasish Mitra, Priyanka Raina, and Boris Murmann. 2023. EMBER: A 100 MHz, 0.86 mm2, Multiple-Bits-per-Cell RRAM Macro in 40 nm CMOS with Compact Peripherals and 1.0 pJ/bit Read Circuitry. In *ESSCIRC 2023- IEEE 49th European Solid State Circuits Conference (ESSCIRC)*. 469–472. https://doi.org/10.1109/ESSCIRC59616.2023.10268807

[28] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, S Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, et al. 2021. Edge AI without compromise: efficient, versatile and accurate neurocomputing in resistive random-access memory. arXiv preprint arXiv:2108.07879.

[29] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, S. Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, Siddharth Joshi, Huaqiang Wu, H. S. Philip Wong, and Gert Cauwenberghs. 2021. Edge AI without Compromise: Efficient, Versatile and Accurate Neurocomputing in Resistive Random-Access Memory. arXiv:2108.07879 [cs.AR] https://arxiv.org/abs/2108.07879

[30] Chenyu Wang, Ge Shi, Fei Qiao, Rubin Lin, Shien Wu, and Zenan Hu. 2023. Research progress in architecture and application of RRAM with computing-in-memory. *Nanoscale Advances* 5, 6 (2023), 1559–1573.

[31] Chen Wang, Huaqiang Wu, Bin Gao, Lingjun Dai, Ning Deng, DC Sekar, Z Lu, M Kellam, G Bronner, and He Qian. 2015. Relaxation effect in RRAM arrays: Demonstration and characteristics. *IEEE Electron Device Letters* 37, 2 (2015), 182–185.

[32] Anjiang Wei, Akash Levy, Pu Yi, Robert M Radway, Priyanka Raina, Subhasish Mitra, and Sara Achour. 2023. PBA: Percentile-Based Level Allocation for Multiple-Bits-Per-Cell RRAM. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[33] Tony F Wu, Binh Q Le, Robert Radway, Andrew Bartolo, William Hwang, Seungbin Jeong, Haitong Li, Pulkit Tandon, Elisa Vianello, Pascal Vivet, et al. 2019. 14.3 A 43pJ/cycle non-volatile microcontroller with 4.7 $\mu$s shutdown/wake-up integrating 2.3-bit/cell resistive RAM and resilience techniques. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 226–228.

[34] Yue Xi, Bin Gao, Jianshi Tang, Xing Mu, Feng Xu, Peng Yao, Xinyi Li, Wenbin Zhang, Meiran Zhao, He Qian, et al. 2020. Impact and quantization of short-term relaxation effect in analog RRAM. In *2020 4th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. IEEE, 1–4.

[35] Yue Xi, Jianshi Tang, Bin Gao, Feng Xu, Xinyi Li, Yuyao Lu, He Qian, and Huaqiang Wu. 2022. The impact of thermal enhance layers on the relaxation effect in analog rram. *IEEE Transactions on Electron Devices* 69, 8 (2022), 4254–4258.

[36] Yachen Xiang, Peng Huang, Yudi Zhao, Meiran Zhao, Bin Gao, Huaqiang Wu, He Qian, Xiaoyan Liu, and Jinfeng Kang. 2019. Impacts of state instability and retention failure of filamentary analog RRAM on the performance of deep neural network. *IEEE Transactions on Electron Devices* 66, 11 (2019), 4517–4522.

[37] Yuan Xie. [n. d.]. RETROSPECTIVE: PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. *Machine learning* 30, 35 ([n. d.]), 40.

[38] Cheng-Xin Xue, Wei-Hao Chen, Je-Syu Liu, Jia-Fang Li, Wei-Yu Lin, Wei-En Lin, Jing-Hong Wang, Wei-Chen Wei, Ting-Wei Chang, Tung-Cheng Chang, et al. 2019. 24.1 A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 388–390.