Best Paper Award, DSN 2019

Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices

Minesh Patel[†] Jeremie S. Kim[‡†] Hasan Hassan[†] Onur Mutlu[†‡]

[†]*ETH Zürich* [‡]*Carnegie Mellon University*

Experimental characterization of DRAM errors is a powerful technique for understanding DRAM behavior and provides valuable insights for improving overall system performance, energy efficiency, and reliability. Unfortunately, recent DRAM technology scaling issues are forcing manufacturers to adopt on-die error-correction codes (ECC), which pose a significant challenge for DRAM error characterization studies by obfuscating raw error distributions using undocumented, proprietary, and opaque error-correction hardware. As we show in this work, errors observed in devices with on-die ECC no longer follow expected, well-studied distributions (e.g., lognormal retention times) but rather depend on the particular ECC scheme used.

In this work, we develop Error-correction INference (EIN), a new statistical inference methodology that overcomes the inability to understand the error characteristics of DRAM devices with on-die ECC. EIN uses maximum a posteriori (MAP) estimation over statistical models that we develop to represent ECC operation to: i) reverse-engineer the ECC scheme and ii) infer the pre-correction error rates given only the post-correction errors. We design and publicly release EINSim, a flexible open-source simulator that can apply EIN to a wide variety of DRAM devices and standards.

We evaluate EIN through the first experimental error-characterization study of DRAM devices with on-die ECC in open literature. Using the data-retention error rates of 232 (82) LPDDR4 devices with (without) on-die ECC across a wide range of temperatures, refresh rates, and test patterns, we show that EIN enables: i) reverse-engineering the on-die ECC scheme, which we find to be a single-error correction Hamming code with (n = 136, k = 128, d = 3), ii) inferring pre-correction error rates given only post-correction errors, and iii) recovering the well-studied pre-correction error distributions that on-die ECC obfuscates.

1. Introduction

DRAM has long since been a crucial component in computing systems primarily due to its low cost-per-bit relative to alternative memory technologies [73, 88, 91, 92]. However, while subsequent technology generations have substantially increased overall DRAM capacity, they have not achieved comparable improvements in performance, energy efficiency, and reliability [12, 32, 73, 91]. This has made DRAM a significant performance and energy bottleneck in modern systems [88, 91].

To address this challenge, researchers propose a wide variety of solutions based on insights and understanding about DRAM behavior gleaned from *system-level DRAM error characterization* studies [5, 10, 12, 15, 24, 27, 33, 34, 40, 45–47, 50–53, 56–59, 61, 70, 72, 79–81, 85, 95–99, 101, 105, 106, 114–121, 124, 125, 129, 134]. These studies deliberately induce errors in a DRAM device by experimentally testing the device at conditions that exacerbate physical DRAM error mechanisms (e.g., charge leakage, circuit interference). The resulting errors directly reflect the effects of the error mechanisms, providing researchers with insight into the physical properties that underlie DRAM operation (e.g., data-retention, circuit timings, data-pattern sensitivity). Researchers can then exploit these insights to develop new mechanisms that improve DRAM and overall system efficiency.

Unfortunately, continued DRAM technology scaling heralds grave reliability concerns going forward primarily due to increasing single-bit error rates that reduce manufacturing yield [28, 37, 49, 73, 82, 85, 86, 92, 93, 106, 114, 115]. While manufacturers traditionally use redundant circuit elements (e.g., rows, columns) to repair manufacturing faults [28, 38, 49, 84, 92, 113], mitigating growing single-cell error rates is no longer tractable using circuit-level redundancy alone [86].

To maintain desired yield targets, DRAM manufacturers have recently supplemented circuit-level redundancy with *on-die error correction codes (on-die ECC)*[1] [49, 86, 92–94]. On-die ECC is *completely invisible to the system* [49, 93]: its implementation, encoding/decoding algorithms, and metadata are all fully contained within the DRAM device and provide no feedback about error detection and/or correction to the rest of the system. On-die ECC is independent of any particular DRAM standard, and JEDEC specifications do not constrain how the on-die ECC mechanism may be designed [44]. Since DRAM manufacturers primarily employ on-die ECC to transparently improve yield, they do not publicly release the ECC implementation details. Therefore, on-die ECC is typically not described in DRAM device datasheets, and neither publications [17, 48, 49, 67, 68, 94] nor whitepapers [41, 86] provide details of the ECC mechanism for a given product.

Unfortunately, on-die ECC has dire implications for DRAM error characterization studies since it censors the true errors that result from physical error mechanisms inherent to DRAM technology. For a device with on-die ECC, we observe only *post-correction* errors, which do not manifest until *pre-correction* error rates exceed the ECC's correction capability. However, the way in which the ECC mechanism transforms a specific uncorrectable error pattern is implementation-defined based on the mechanism's design, which is *undocumented*, *proprietary*, *opaque*, and possibly *unique* per product. Thus, on-die ECC effectively obfuscates the pre-correction errors such that they *cannot* be measured simply by studying post-correction errors without knowing the ECC scheme.

Figure 1 demonstrates the differences in the *observed* data-retention bit error rate (BER) (y-axis) for different on-die ECC schemes (explained in Section 3.3) given the same *pre-correction*

[1] Also known as *in-DRAM ECC* and *integrated ECC*.

BER (x-axis). We generate this data in simulation using EINSim, which is described in detail in Section 5. We see that the observed error rates are *dependent* on the particular ECC scheme used, and without knowledge of which ECC scheme is used in a given device, there is no easy way to tie the *observed* error rates to the *pre-correction* error rates.

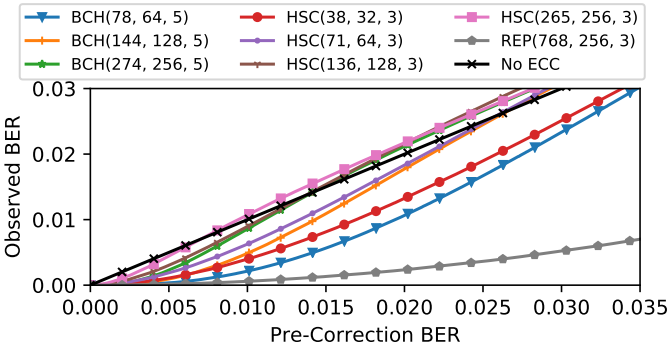

Figure 1: Observed vs. pre-correction data-retention bit error rate (BER) for various ECC schemes (color) and no ECC (black) assuming 256 data bits written with `RANDOM` data (simulated).

This means that post-correction errors may not follow expected, well-studied distributions based on physical error mechanisms (e.g., exponential temperature dependence of charge leakage rates [4, 30, 79], lognormal retention-time distributions [30, 76, 80, 98]) but rather device-architecture-specific shapes that cannot be reliably compared with those from a device with a different ECC scheme. We discuss and experimentally demonstrate the implications of this observation in Sections 2 and 8, respectively.

Thus, on-die ECC effectively *precludes* studying DRAM error mechanisms, motivating the need for a DRAM error characterization methodology that isolates the effects of intrinsic DRAM behavior from those of the ECC mechanism used in a particular device. To this end, **our goal in this work** is to overcome the barrier that on-die ECC presents against understanding DRAM behavior in modern devices with on-die ECC. To achieve this goal, we develop Error-correction INference (EIN), a statistical inference methodology that uses maximum a posteriori (MAP) estimation to 1) reverse-engineer the ECC scheme and 2) infer the *pre-correction* error rates from only the observed *post-correction* errors. We follow a methodical four-step process:

First, we tackle the unique reverse-engineering problem of determining the on-die ECC scheme *without* any visibility into the error-correction algorithm, the redundant data, or the locations of pre-correction errors. Our approach is based on the **key idea** that even though ECC obfuscates the exact locations of the pre-correction errors, we can leverage known statistical properties of pre-correction error distributions (e.g., uniform-randomness [5, 57, 98, 112]) in order to disambiguate the effects of different ECC schemes (Section 4).

We develop statistical models to represent how a given pre-correction error distribution will be transformed by an arbitrary ECC scheme (Section 4.1). Our models are parameterized by *i*) the desired ECC scheme and *ii*) statistical properties of the pre-correction error distribution. We then formulate the reverse-engineering problem as a maximum a posteriori (MAP) estimation of the most likely model given experimental data from real devices (Section 4.4).

Second, in order to compute several expressions in our statistical models that are difficult to evaluate analytically, we develop EINSim [1], a flexible open-source simulator that numerically estimates the error-detection, -correction, and -miscorrection effects of arbitrary ECC schemes for different pre-correction error distributions (Section 5). EINSim models the lifetime of a given ECC dataword through the encoding, error injection, and decoding processes faithful to how these steps would occur in a real device (Section 5.1). To ensure that EINSim is applicable to a wide range of DRAM devices and standards, we design EINSim to be modular and easily extensible to additional error mechanisms and distributions.

Third, we perform the *first* experimental study of DRAM devices with on-die ECC in open literature and demonstrate how EIN infers both: *i*) the on-die ECC scheme and *ii*) the pre-correction error rates. We study the data-retention characteristics of 232 (82) state-of-the-art LPDDR4 DRAM devices with (without) on-die ECC from one (three) major DRAM manufacturers across a wide variety of temperatures, refresh rates, and test patterns. To accurately model pre-correction errors in EINSim, we first reverse-engineer:

- The layout and dimensions of internal DRAM cell arrays.
- The locations and frequency distribution of redundant DRAM rows used for post-manufacturing repair.

Applying EIN to data from devices with on-die ECC, we:

- Find that the on-die ECC scheme is a single-error correction Hamming code [31] with (n = 136, k = 128, d = 3).
- Show that EIN can infer *pre-correction* error rates given only *post-correction* errors.

Fourth, we demonstrate EIN's usefulness by providing a proof-of-concept experimental characterization study of the data-retention error rates for the DRAM devices with on-die ECC. We test across different refresh intervals and temperatures to show that EIN effectively enables inferring the pre-correction error rates, which, unlike the ECC-obfuscated post-correction error rates, follow known shapes that result from well-studied device-independent error mechanisms.

2. Motivation

EIN allows researchers to more holistically study the reliability characteristics of DRAM devices with on-die ECC by exposing the pre-correction error rates beneath the observed post-correction errors. This enables researchers to propose new ideas based on a more general understanding of DRAM devices. To demonstrate how EIN may be useful, we provide: 1) several examples of studies and mechanisms that EIN enables and 2) a discussion about the implications of continued technology scaling for future error characterization studies.

2.1. Example Applications

We provide several examples of potential studies and mechanisms that are enabled by knowing pre-correction error rates, which on-die ECC masks and EIN reveals:

- *Runtime Error Rate Optimization*: A mechanism that intelligently adjusts operating timings/voltage/frequency to

meet dynamically changing system performance/energy/reliability targets (e.g., Voltron [15], AL-DRAM [70], AVA-DRAM [71], DIVA-DRAM [72]) typically needs to profile the error characteristics of a device for runtime decision-making. If the particular ECC scheme is known (e.g., using EIN), such a mechanism can leverage device-independent DRAM error models for decision-making and quickly interpolating or extrapolating "safe" operating points rather than having to: (1) use complex, likely non-parametric, device-specific models for each supported ECC scheme or (2) characterize each device across its entire region of operation.

- *Device Comparison Studies*: EIN enables fair comparisons of DRAM error characteristics between devices with different (and without) on-die ECC mechanisms. This is useful for studying the evolution of error characteristics over time, which provides insight into the state of the industry and future technology trends. With DRAM error rates continuing to worsen (discussed in Section 2.2), such studies can help predict how much worse future devices may be and how well current/future error-mitigation mechanisms will cope.
- *Reverse-Engineering Other ECCs*: As we discuss in Section 5.6, EIN is applicable to other systems (e.g., rank-level ECC, Flash memory) whose ECC schemes are typically also proprietary. Reverse-engineering their ECC schemes can be useful for various reasons [7, 19, 21, 26, 131], including failure analysis, security evaluation, forensic analysis, patent infringement, and competitive analysis. For these systems, EIN may provide a way to reverse-engineer the ECC scheme without requiring hardware intrusion or internal access to the ECC mechanism as typically required by previous approaches [19, 122, 123, 131] (discussed in Section 9).

We hope that future work will use EIN well beyond these use cases and will develop new characterization-driven understanding of devices with on-die ECC.

2.2. Applicability to Future Devices

Despite its energy and reliability benefits, on-die ECC does not fundamentally prevent error rates from increasing. Therefore, future DRAM devices may require *stronger* error-mitigation solutions, *further obfuscating* pre-correction error rates and making error characterization studies even more difficult.

Similarly, other memory technologies (e.g., Flash [7, 8], STT-MRAM [42, 66, 138], PCM [69, 100, 107, 133], Racetrack [135], RRAM [132]) suffer from ongoing reliability concerns, and characterizing their error mechanisms requires surmounting any error-mitigation techniques they use. EIN takes a first step towards enabling a holistic understanding of devices whose error characteristics are not directly visible, and we hope that future work leverages this opportunity to develop new mechanisms to tackle the reliability challenges that lie ahead.

3. Background

We provide the necessary background on DRAM operation and error-correction codes (ECC) for understanding our motivation, experimentation, and analysis. For further detail, we refer the reader to prior works on DRAM optimization [11–14, 32, 33, 52, 57, 61–63, 70–74, 108–110, 136] and coding theory [9, 20, 31, 39, 49, 82, 83, 93, 102, 103, 130].

3.1. DRAM Organization

DRAM is organized in a hierarchy of two-dimensional arrays as shown in Figure 2. Figure 2a illustrates a single DRAM cell and its associated peripheral circuitry. Each cell encodes one bit of data using the charge level in its *capacitor*. A *true-cell* encodes data '1' as fully charged (i.e., V_{DD}) and data '0' as fully discharged (i.e., V_{SS}), whereas an *anti-cell* uses the opposite encoding. The cell is accessed by driving the *wordline*, which enables the *access transistor* and connects the *bitline* to the cell.

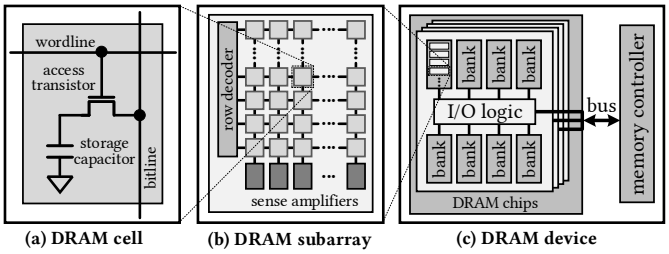

Figure 2: DRAM organization.

Figure 2b shows how cells are packed in a grid to form a *subarray* [62]. A wordline spans a *row* of DRAM cells, typically one or more KiB long, and is the minimum granularity of DRAM array operation. Upon a DRAM access, the *row decoder* drives the corresponding wordline, thus *activating* the row. This allows the charge stored in each cell along the activated row to be sensed by the *sense amplifiers* connected to the cells' respective bitlines. Because cells in a *column* share a single bitline, a subarray may have only one row active at a time [62].

Multiple subarrays are aggregated to form a larger array referred to as a *bank*, and banks are in turn combined to form a *chip* as depicted in Figure 2c. I/O circuitry within each chip interfaces the individual banks with the external DRAM bus. A set of DRAM chips that share a common bus is known as a *rank*, and one or more ranks may be combined via rank selection signals to form a single DRAM *channel*. The DRAM bus is connected to a *memory controller*, which typically resides within the processor die. Each DRAM access transfers one *burst* of data that consists of multiple bus-width *beats*. For LPDDR4 DRAM, bursts are typically 32B or 64B long, and each beat is 16 bits long [44].

3.2. DRAM Timings and Errors

The memory controller interfaces with DRAM according to manufacturer-specified *timing parameters*,[2] which guarantee correct DRAM behavior by providing enough time in between DRAM commands for internal DRAM circuitry to stabilize. Our work primarily deals with with DRAM refresh timings (Section 3.2.1) and the data-retention errors that result from violating refresh timing specifications (Section 3.2.2).

3.2.1. DRAM Refresh Timing. DRAM cell capacitors inherently lose charge over time [79, 80, 104], potentially resulting in data corruption. A cell's *retention time* defines how long it can reliably store data and typically varies between cells from milliseconds to many hours [33,49,51,59,70,76,79,80,98,99,124]. To prevent data loss, the memory controller regularly *refreshes* the

[2]We encourage the interested reader to refer to the JEDEC specification [44] for an exhaustive list of all available parameters and their usage.

entire DRAM memory using periodic *REF* commands, which are scheduled according to a timing parameter called the *refresh window* (t_{REFW}). t_{REFW} defines the maximum amount of time allowed between consecutive refresh operations to a given DRAM cell. In our work, we experimentally test LPDDR4 DRAM devices (Sections 6, 7, and 8), for which t_{REFW} is 32ms at typical operating conditions [44].

3.2.2. Violating Recommended Timings. We can induce errors[3] in real DRAM devices by deliberately violating manufacturer-recommended timings. The resulting error distributions allow us to: 1) reverse-engineer various proprietary DRAM microarchitectural characteristics [45, 70, 72] and 2) understand the behavior of different DRAM error mechanisms (e.g., charge leakage [30, 79], circuit crosstalk [52, 61, 111]).

By increasing t_{REFW}, we observe *data-retention errors* in certain cells with higher charge leakage rates [30, 49, 59, 80, 124]. The quantity and locations of these errors depend on *i*) the data pattern programmed into cells, *ii*) the layout of true- and anti-cells in DRAM [65, 79], and *iii*) environmental factors such as operating temperature and voltage [15, 30, 51–54, 61, 75, 76, 79, 80, 98]. Section 4.3 discusses the statistical characteristics of data-retention errors in greater depth.

3.3. Block Error-Correction Codes

Block coding enables data communication over a noisy channel by breaking the data stream into *datawords* of length k, where each element of the dataword is a *symbol* representing q bits of data. During *encoding*, the *ECC encoder* maps each dataword to a single *codeword* of length n using $n - k$ *redundant symbols*. Each symbol is a function (e.g., xor-reduce) of a subset of the data symbols such that an error will cause one or more of these functions to evaluate incorrectly. Encoding results in q^k *valid codewords* out of q^n possible n-symbol words. Upon receiving an n-symbol word that may contain erroneous symbol(s), the *ECC decoder* attempts to determine the originally transmitted dataword using a *decoding algorithm*.

As a demonstrative example, we consider a common decoding algorithm for binary (i.e., $q = 2$) block codes known as *maximum-likelihood decoding*, which uses Hamming distance as a metric to find the closest valid codeword to a received word. Using this approach, the *error-correction capability*, or t, is defined by the *minimum Hamming distance*, or d, between any two valid codewords in the space of all valid codewords. With $d = 2$, a single-symbol error can always be detected but not always corrected since there may exist two valid codewords equidistant from the received word. In general, the error-correction capability can be computed using the relationship $t = \lfloor \frac{d-1}{2} \rfloor$, which shows that a minimum Hamming distance of at least 3 is necessary for single-symbol correction and 5 for double-symbol correction.

When faced with more errors than the code can correct, the decoding result is *implementation-defined* based on the exact circuitry used to implement the encoding and decoding algorithms. This is because a code designer has complete freedom to choose the precise functions that map data symbols to each redundant symbol, and the same errors induced in two different code implementations can result in two different post-correction words. In each implementation, the decoding logic may *i*) manage to correct one or more actual errors, *ii*) mistakenly do nothing, or *iii*) "miscorrect" a symbol that did *not* have an error, effectively *exacerbating* the number of errors in the decoding result.

Throughout this work, we follow a commonly used notation for ECC block codes, in which a tuple (n, k, d) describes the length of the codeword (n), the length of the dataword (k), and the minimum Hamming distance (d), respectively. This allows us to concisely express the type and strength of a block code. However, certain codes are also well-known by name (e.g., Repetition (REP) [22], Hamming Single-Error Correction (HSC) [31], Bose-Chaudhuri-Hocquenghem (BCH) [6, 36]), and we will use these names where appropriate.

4. Statistically Modeling DRAM and ECC

We begin by formalizing the relationship between pre- and post-correction error distributions and expressing reverse-engineering as a maximum a posteriori (MAP) estimation problem. Our approach is grounded on the key idea that pre-correction errors arise from physical error mechanisms with known statistical properties, and because different ECC schemes transform these distributions in different ways, we can use what we know about both the pre- and post-correction error distributions to disambiguate different ECC schemes. This section provides a step-by-step derivation of EIN, the statistical inference methodology we propose in this work.

4.1. Statistically Modeling Error Correction Codes

Consider an ECC mechanism implementing an (n, k, d) binary block code as illustrated in Figure 3. The ECC *encoding* algorithm $f_{enc,\ ECC}$ transforms a dataword w out of the set of all possible datawords $\mathcal{W} = \mathbb{Z}_2^k$ into a valid codeword c out of the set of all possible valid codewords $\mathcal{C} \subset \mathbb{Z}_2^n$. Likewise, the *decoding* algorithm $f_{dec,\ ECC}$ transforms a codeword c' (potentially invalid due to errors) out of the set of all possible codewords $\mathcal{C}' = \mathbb{Z}_2^n$ into a corrected dataword w' out of the set of all possible corrected datawords $\mathcal{W}' = \mathbb{Z}_2^k$.

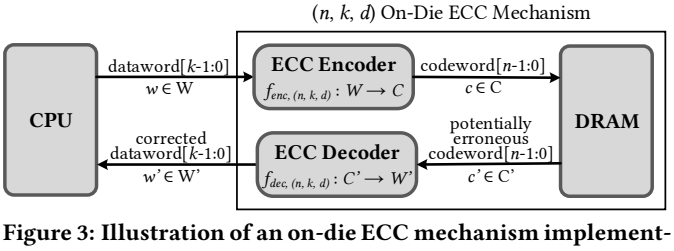

Figure 3: Illustration of an on-die ECC mechanism implementing an (n, k, d) binary block code.

$f_{dec,\ ECC}$ can be thought of as a deterministic mapping[4] from the finite set of inputs $\mathcal{C}'$ to a finite set of outputs $\mathcal{W}'$:

$$f_{dec,\ ECC} \ : \ \mathcal{C}' \mapsto \mathcal{W}' \tag{1}$$

[3] According to the IEEE TCRTS [2], a *fault* is a defect inherent to a system, an *error* is a discrepancy between intended and actual behavior, and a *failure* is an *observed* instance of incorrect behavior. We conform to this terminology throughout this manuscript.

[4] While non-deterministic encoding/decoding algorithms exist, they are typically not used with the simple ECCs found in DRAM. If more complex ECCs must be considered (e.g., LDPC [7, 20]), our models can be extended to treat the encoding/decoding functions as probabilistic transformations themselves.

This means that for a particular ECC scheme f_i, the probability of observing output w' is determined by the probabilities that its corresponding inputs $\{c'_j \in \mathcal{C}',\ \forall j : f_{dec,\ i}(c'_j) = w'\}$ occur:

$$\mathrm{P}_{f_i}[w'] = \sum_{\forall j\ :\ f_{dec,\ i}(c'_j)=w'} \mathrm{P}[c'_j] \tag{2}$$

From this perspective, if we know both 1) the ECC scheme f_i and 2) the frequency distribution of all possible *input* values c_k, we can calculate the corresponding distribution of all possible *output* values.[5] Inverting this relationship, if we *experimentally measure* the frequency distribution of output values from a real device, we can determine the probability of having made such observations given 1) a suspected ECC scheme and 2) an expected frequency distribution of all possible inputs. Section 4.3 describes what we know about pre-correction error distributions and how we leverage this knowledge to disambiguate different suspected ECC schemes.

4.2. Experimental Observables

Solving Equation 2 requires measuring the relative frequency distribution of post-correction datawords (i.e., w'_i). For example, if we use 64-bit datawords, we have 2^{64} *unique* datawords. Unfortunately, a single DRAM device has on the order of *millions* of datawords, which is nowhere near enough to obtain a representative sample of the full distribution.

Instead, we divide $\mathcal{W}'$ into $N+1$ subsets, $\mathcal{W}'_n$, which each comprise all possible datawords with $n \in [0, N]$ errors. Using this approach, a relative frequency distribution of the $\mathcal{W}'_n$ contains only $N+1$ categories, and even a single DRAM device contains more than enough samples to obtain a representative distribution. Experimentally, measuring the number of errors in each dataword simply requires counting errors.[6] We can then rewrite Equation 2 in terms of the subsets $\mathcal{W}'_n$:

$$\mathrm{P}_{f_i}[w' \in \mathcal{W}'_n] = \sum_{\forall j\ :\ f_{dec,\ i}(c'_j)\in\mathcal{W}'_n} \mathrm{P}_{f_i}[c'_j] \tag{3}$$

Unfortunately, this approach requires knowing the exact layout of ECC words in memory. This may be difficult since multiple bits are read/written together at the granularity of a *burst* (Section 3.1), and each burst may contain one or more ECC words with an arbitrary bit-to-ECC-word mapping.

To circumvent this problem, we instead consider the probability of observing n errors *per burst*, where each burst comprises dataword(s) from one or more ECC schemes. Mathematically, the total number of errors in a burst is the sum of the individual per-dataword error counts and is computed by convolving the per-dataword error-count distributions. Counting errors at burst-granularity is independent of the layout of ECC words within a burst assuming that ECC words are contained within burst boundaries so that bursts can be read/written independently. However, if a different design is suspected, even longer words (e.g., multiple bursts) may be used as necessary.

[5]Note that since ECC decoding is generally *not* injective (i.e., multiple codewords may map to a single decoded dataword), we cannot determine exactly which input produced an observed output.

[6]There is no fundamental reason for this choice beyond experimental convenience; if another choice is made, our analysis still holds but will need to be modified to accommodate the new choice of $\mathcal{W}'_n$.

4.3. Statistically Modeling DRAM Errors

To estimate the relative frequencies of the *pre-correction* codewords $c'_j \in \mathcal{C}'$, we exploit the fact that errors arise from physical phenomena that follow well-behaved statistical distributions. Throughout this work, we focus on data-retention error distributions since they are well-studied and are easily reproduced in experiment. However, EIN is applicable to *any* experimentally-reproducible error distribution whose statistical properties are well-understood (e.g., reduced activation-latency [12, 56–58, 70, 72], reduced precharge-latency [12, 117, 118], reduced voltage [15], RowHammer [61, 89, 90]).

As described in Section 3.2.2, data-retention errors occur when a charged cell capacitor leaks enough charge to lose its stored value. This represents a "1" to "0" error for a charged true-cell (i.e., programmed with data "1"), and vice-versa for an anti-cell [19, 61, 79]. Due to random manufacturing-time variations [23, 59, 61, 70, 71, 76, 137], certain cells are more prone to data-retention errors than others [30, 49, 59, 79, 80, 124]. Furthermore, absolute data-retention error rates depend on operating conditions such as refresh timings, data patterns, ambient temperature, and supply voltage. Through extensive error characterization studies, prior works find that, for a fixed set of testing conditions (e.g., t_{REFW}, temperature), data-retention errors show no discernible spatial patterns [5, 30, 80, 112, 124] and can be realistically modeled as uniform-randomly distributed [5, 57, 112] independent events [112].

To model an arbitrary pre-correction error distribution in our analysis, we introduce an abstract model parameter θ that encapsulates all state necessary to describe the distribution. In general, θ is a *set* of two key types of parameters: *i*) experimental testing parameters (e.g., data pattern, timing parameters, temperature) and *ii*) device microarchitectural characteristics (e.g., spatial layout of true- and anti-cells). We incorporate θ into our analysis as a dependency to the terms in Equation 3:

$$\mathrm{P}_{f_i,\theta}[w' \in \mathcal{W}'_n] = \sum_{\forall j\ :\ f_{dec,\ i}(c'_j)\in\mathcal{W}'_n} \mathrm{P}_{f_i,\theta}[c'_j] \tag{4}$$

Ideally, *all* of the parameters that comprise θ are known at testing time. Unfortunately, experiments are often imperfect, and internal device characteristics are difficult to obtain without proprietary knowledge or laborious reverse-engineering. If such parameters are unknown, we can infer them alongside the unknown ECC scheme.[7]

In this work, we model data-retention errors as uniform-random, independent events among cells programmed to the "charged" state with a fixed probability determined by testing conditions. θ then encapsulates *i*) the single-bit error probability, called the *raw bit error rate (RBER)*, *ii*) the programmed data pattern, and *iii*) the spatial layout of true-/anti-cells.

Unfortunately, evaluating Equation 4 analytically is difficult even for data-retention errors due to the complexity of the interactions between the ECC scheme and the parameters encompassed by θ. Instead, we numerically estimate the solution to Equation 4 using Monte-Carlo simulation as described in Sec-

[7]While we could lump the unknown ECC scheme into θ as an unknown microarchitectural characteristic, we keep it logically separate since θ represents what we already understand about DRAM devices, and the unknown ECC scheme represents what we do not.

tion 5. This approach allows our analysis to flexibly take into account arbitrarily complex model parameters (e.g., detailed microarchitectural characteristics, nontrivial error models).

4.4. Inferring the Model Parameters

We now formulate the reverse-engineering task as a maximum a posteriori (MAP) estimation problem over a set $\mathcal{F}$ of hand-selected ECC schemes that are either directly mentioned in context with on-die ECC (HSC(71, 64, 3) [41, 43, 93] and HSC(136, 128, 3) [17, 48, 67, 68, 86]) or are used as demonstrative examples of applying our methodology to devices with stronger and/or more complicated codes (e.g., BCH(n, k, d), HSC(n, k, d), REP(3, 1, 3)). Note that we also take into account implementation details of each of these schemes (e.g., systematic vs. non-systematic encodings) using our simulation infrastructure as we describe in Section 5.3.

To reverse-engineer the unknown ECC scheme f_{unknown}, we start by expressing it as the *most likely* ECC scheme out of all possible schemes $f_i \in \mathcal{F}$ given a set of observations $\mathcal{O}$:

$$f_{\text{unknown}} = \underset{f_i}{\text{argmax}}(\text{P}[f_i \mid \mathcal{O}]) \tag{5}$$

Unfortunately, we cannot directly evaluate Equation 5 since our observations $\mathcal{O}$ are measured from a device with a *fixed* ECC scheme. Instead, we use the Bayes theorem to express Equation 5 in terms of the probability of obtaining measurements $\mathcal{O}$ given an *arbitrary* ECC scheme f_i, which we can calculate using the relationship in Equation 3. This yields:

$$\begin{aligned} f_{\text{unknown}} &= \underset{f_i}{\text{argmax}} \left(\frac{\text{P}[\mathcal{O} \mid f_i]\text{P}[f_i]}{\text{P}[\mathcal{O}]} \right) \\ &= \underset{f_i}{\text{argmax}} \left(\text{P}[\mathcal{O} \mid f_i]\text{P}[f_i] \right) \end{aligned} \tag{6}$$

Note that we ignore the denominator (i.e., the marginal likelihood) in Equation 6 because it is a fixed scale factor independent of f_i and does not affect the maximization result.

We assume a uniformly-distributed prior (i.e., P[f_i]) given that we cannot guarantee anything about the on-die ECC implementation. By restricting our analysis to only the aforementioned ECC schemes, we already exclude any schemes that we consider to be unrealistic. In principle, we could assign greater or lower probability mass to schemes that have been mentioned in prior work or that are exceedingly expensive, respectively, but we choose not to do so because *i*) we cannot guarantee that the devices we test are similar to those mentioned in prior work, and *ii*) we want to demonstrate the power of our methodology without biasing the results towards any particular ECC schemes.

The likelihood function (i.e., P[$\mathcal{O} \mid f_i$]) incorporates the experimental data we obtain from real devices. As we show in Section 4.2, our measurements provide us with the probability of observing an n-bit error in each of j independent DRAM bursts. Defining N as a random variable representing the number of erroneous bits observed in a single burst and assuming observations are independent events (validated in Section 4.3), we rewrite the likelihood function as:

$$\text{P}[\mathcal{O} \mid f_i] = \text{P}_{f_i,\theta}\left[\bigcap_{j=0}^{j_{max}} N = n_j\right] = \prod_{j=0}^{j_{max}} \text{P}_{f_i,\theta}[N = n_j] \tag{7}$$

This is essentially a multinomial probability mass function (PMF) evaluated at $\mathcal{O}$, where each probability mass is computed using Equation 4. Unfortunately, as described in Section 4.3, the model parameter θ encapsulates the *pre-correction* error rate, which we do not know and cannot measure post-correction. Therefore, for each ECC scheme f_i, we first maximize the likelihood distribution over θ:

$$\text{P}[\mathcal{O} \mid f_i] = \max_{\theta} \left(\prod_{j=0}^{j_{max}} \text{P}_{f_i,\theta}[N = n_j] \right) \tag{8}$$

Inserting the result of Equation 8 into our original optimization objective (Equation 6), we obtain the final objective function to optimize in order to reverse-engineer the ECC scheme f_{unknown} used in our devices:

$$f_{\text{unknown}} = \underset{f_i}{\text{argmax}} \left(\max_{\theta} \left(\prod_{j=0}^{j_{max}} \text{P}_{f_i,\theta}[N = n_j] \right) \text{P}[f_i] \right) \tag{9}$$

where the inner product term is calculated using Equation 4.

After the ECC scheme is reverse-engineered, we can repeatedly apply Equation 8 to solve for θ across many different experiments (i.e., observations). Since θ represents all parameters necessary to describe the pre-correction error distribution (described in Section 4.3), this is equivalent to reverse-engineering the pre-correction error rate. With the ECC scheme known as f_{known}, Equation 8 simplifies to:

$$\theta_{\text{unknown}} = \underset{\theta}{\text{argmax}} \left(\prod_{j=0}^{j_{max}} \text{P}_{f_{\text{known}},\theta}[N = n_j] \right) \tag{10}$$

With Equations 9 and 10, we can reverse-engineer both *i*) the ECC scheme and *ii*) the pre-correction error rates from observed post-correction errors for *any* DRAM device whose error distributions are obscured by ECC. In Section 7.3, we experimentally demonstrate how to apply Equation 9 to real devices with on-die ECC.

5. Simulation Methodology

To apply EIN to data from real devices, we develop and publicly release EINSim [1], a flexible open-source C++-based simulator that models the life of a dataword through the entire ECC encoding/decoding process. EINSim accounts for different ECC implementations and pre-correction error characteristics to ensure that EIN is applicable to a wide variety of DRAM devices and standards. This section describes EINSim's extensible design and explains how EINSim can be used to solve the optimization problems formulated in Section 4.

5.1. EINSim High-Level Architecture

Figure 4 shows a high-level diagram of the logical flow of data through EINSim's different components. To model a DRAM experiment, we simulate many individual burst-length accesses that each access a different group of cells. Each burst simulates an experimental measurement, yielding a distribution of measured values across all simulated bursts. We describe the function of each simulator component.

❶ **Word generator** constructs a bitvector using commonly tested data patterns (e.g., `0xFF`, `0xAA`, `RANDOM`) [3, 51, 52, 72, 79, 98, 128], simulating the data written to DRAM.

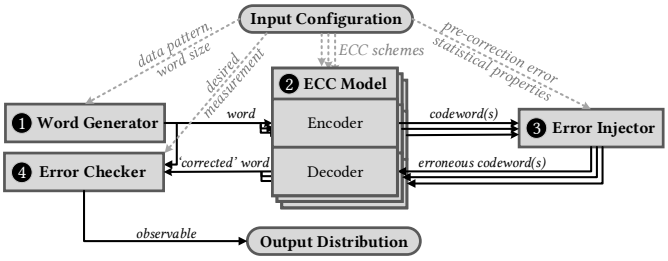

Figure 4: High-level block diagram showing the logical flow of data through the different components in our simulator.

❷ **ECC model** encompasses an ECC implementation, including the encoding/decoding algorithms and implementation details such as systematic vs. non-systematic encodings. Because a single word from the word generator may comprise multiple ECC datawords, EINSim provides a configurable mapping for decomposing the word into ECC datawords.

❸ **Error injector** injects errors into a given codeword according to a configurable error distribution. We implement support for data-retention errors as described in Section 4.3. We provide configurable parameters for the spatial distribution of true-/anti-cells (e.g., alternating per row) and the single-bit probability of failure (i.e., RBER). Errors are injected uniform-randomly across each bit that can fail (i.e., each "charged" cell per the chosen true-/anti-cell layout and data pattern) using a Bernoulli distribution with p equal to the desired RBER normalized by the ratio of all cells that can fail, which ensures that the simulated error rate meets the target RBER on average.

❹ **Error checker** takes the pre- and post-correction words as inputs and calculates a user-defined measurement (e.g., total number of bit-flips). This corresponds to an experimental observable as explained in Section 4.2.

5.2. EINSim Validation

We validate EINSim using a combination of manual and automatic unit tests. For the ECC model, we 1) provide tests for detecting/correcting the right amount of errors (exhaustively/sample-based for short/long codes); 2) hand-verify the inputs/outputs of encoders/decoders where reasonable; 3) hand-validate the generator/parity-check matrices and/or code generator polynomials against tables of known values (e.g., [18]); and 4) validate the minimum distance and weight distributions of codewords. Due to the simplicity of how we model the true-/anti-cell layout and data-retention errors, we validate the error-injection correctness by 1) manual inspection and 2) using summary statistics (e.g., distribution of errors across many simulated bursts).

5.3. Applying EINSim to Experimental Data

To analyze data taken from a real experiment, we configure the simulation parameters to match the experiment and simulate enough read accesses (e.g., $>10^5$) to allow the distribution of simulated measurements to numerically estimate the real experimental measurements. This approach effectively solves Equation 4 through Monte-Carlo simulation for any model parameters $\{f_i, \theta\}$ that can be simulated using EINSim.

Figure 1 in Section 1 provides several examples of evaluating Equation 4 across a wide range of model parameters $\{f_i, \theta\}$ using a 256-bit input word programmed with a `RANDOM` data pattern. The X-axis shows the *pre-correction* bit error rate (BER), i.e., the RBER component of θ, and the Y-axis shows the *observed* BER, which is computed by taking an expectation value over the distribution resulting from solving Equation 4. Curves represent different ECC schemes f_i, and each data point represents one simulation of 10^6 words, subdividing each word into multiple ECC datawords as necessary.

We see that each ECC scheme transforms the pre-correction error rate differently. For example, stronger codes (e.g., REP(768, 256, 3), BCH(78, 64, 5)) dramatically decrease the observed BER, whereas weaker codes (e.g., HSC(265, 256, 3)) have a relatively small effect. Interestingly, we see that many of the codes actually *exacerbate* the error rate at high enough pre-correction error rates because, on average, the decoder mistakenly "corrects" bits without errors more often than not. These examples demonstrate that different ECC schemes have different effects on the pre-correction error distribution, and Equation 9 exploits these differences to disambiguate schemes.

5.4. Inferring the Model Parameters

To infer the model parameters f and θ, which represent the ECC scheme and pre-correction error distribution characteristics, respectively, we need to perform the optimization given by Equation 9. We do this using a grid search across f and θ, simulating 10^4 uniformly-spaced error rates for each of several different ECC schemes, data patterns and true-/anti-cell layouts. While a denser grid may improve precision, this configuration sufficiently differentiates the models we analyze (Section 7.3).

The solutions to Equation 9 are the inferred ECC scheme and pre-correction error distribution characteristics that best explain the experimental observations. From there, we can use Equation 10 evaluated with the known ECC scheme in order to determine θ for any *additional* experiments that we run (e.g., different error rates).

5.5. Inference Accuracy

MAP estimation rigorously selects between known models and inherently can neither confirm nor deny whether the MAP estimate is the "real" answer. We identify this as a limitation of EIN in Section 5.8. However, in the event that a device uses a scheme that is not considered in the MAP estimation, it would be evident when testing across different experimental conditions and error rates since it is unlikely that any of the chosen ECC schemes would be the single maximum-a-posteriori scheme (i.e., best explaining the observed data) across all experiments.

We can also use confidence intervals to gauge the error in each MAP estimate. This requires repeating the MAP estimation over N bootstrap samples [25] taken from the observed data $\mathcal{O}$. The min/max or $5^{\text{th}}/95^{\text{th}}$ percentiles are typically taken to be the confidence bounds.

5.6. Applying EIN to Other Systems

EIN can be extended to any ECC-protected communication channel provided that we can induce uncorrectable errors whose pre-correction spatial distribution follows some known property (e.g., uniform-randomness). Examples include, but are not limited to, DRAM rank-level (i.e., DRAM-controller-

level) ECC and other memory technologies (e.g., SRAM, Flash, Phase-Change Memory).

5.7. Applying EIN to Data from Prior Studies

EIN is applicable to data presented in a prior study if the study supplies enough information to solve Equation 10. This requires that the study provides both: 1) the pre-correction error characteristics, either directly as statistical distributions or implicitly through the experimental methodology (e.g., device model number, tested data patterns) and 2) the distribution of errors amongst post-correction words as discussed in Section 4.2. If these are known, EIN can infer both the ECC scheme and the pre-correction error rates from the given data.

5.8. Limitations of EIN

EIN has three main limitations. However, in practice, these limitations do not hurt its usability since both DRAM and ECC design are mature and well-studied topics. We discuss each limitation individually:

1) *Cannot guarantee success or failure.* As described in Section 5.5, MAP estimation cannot guarantee whether the correct solution has (not) been found. However, Section 5.5 describes how testing across different operating conditions and using confidence intervals helps mitigate this limitation.
2) *Requires knowledge and control of errors.* Using EIN requires *i*) knowing statistical properties of the spatial distribution of pre-correction errors, and *ii*) the ability to induce uncorrectable errors. Fortunately, EIN can use any one of the many well-studied, easily-manipulated error mechanisms that are fundamental to DRAM technology (e.g., data retention, RowHammer, reduced-latency access; see Section 4.3). Such mechanisms are unlikely to change dramatically for future devices (e.g., retention errors are modeled similarly across decades of DRAM technologies [16, 29, 30, 35, 64, 77, 87, 127]), which means that EIN will likely continue to be applicable.
3) *Cannot identify bit-exact error locations.* While EIN infers pre-correction error *rates*, it cannot determine the *bit-exact locations* of pre-correction errors. Unfortunately, since multiple erroneous codewords may map to each visible dataword, we are not aware of a way to infer error locations without insight into the exact ECC implementation (e.g., algorithms, redundant data). However, inferring error rates is sufficient to study aggregate distributions, and we leave error localization to future work.

6. Experimental Setup

We experimentally characterize 232 LPDDR4 [44] DRAM devices *with* on-die ECC from a single major DRAM manufacturer that we cannot disclose for confidentiality reasons. For comparison purposes, we test 82 LPDDR4 DRAM devices of the previous technology generation *without* on-die ECC from across three major DRAM manufacturers. Given that DRAM manufacturers provide neither: *i*) non-ECC counterparts of devices with on-die ECC nor *ii*) a mechanism by which to disable on-die ECC, the older-generation devices provide our closest point of comparison.

We perform all testing using a home-grown infrastructure that provides precise control over DRAM timing parameters, bus commands, and bus addresses. Our infrastructure provides reliable ambient temperature control between 40°C - 55°C with a tolerance of of ±1°C. To improve local temperature stability for each DRAM device throughout testing, a local heating source maintains DRAM at 15°C above the ambient temperature at all times, providing an effective DRAM temperature testing range of 55°C - 70°C.

7. Experimentally Inferring On-Die ECC and Pre-Correction Error Rates Using EIN

In this section, we apply EIN to infer the *i*) on-die ECC scheme and *ii*) pre-correction error rates of real devices with on-die ECC. Before doing so, we validate our uniform-random statistical model for pre-correction errors and determine the layout of true-/anti-cells to accurately model the pre-correction error distribution of the devices that we test.

7.1. Validating Uniform-Random Retention Errors

Our model for data-retention errors (Section 4.3) treats errors as independent, uniform-randomly distributed events based on observations made in several prior works [5, 30, 57, 80, 112, 124]. For such errors, the total number of errors X in each fixed-length n-bit region of DRAM follows a binomial distribution [5, 98, 124] parameterized by the RBER R:

$$\mathrm{P}[X = x \mid R] = \binom{n}{x} R^x (1-R)^{n-x} \tag{11}$$

Before demonstrating the use of EIN, we first validate that the independent, uniform-random data-retention error model holds for the devices that we test by comparing experimentally-measured error distributions to the *expected* distributions. Figure 5 shows both the expected and experimental probabilities of observing an X-bit error in a single 256-bit word throughout DRAM at fixed operating conditions of t_{REFW} = 20s and 60°C for a single representative DRAM device *without* on-die ECC.

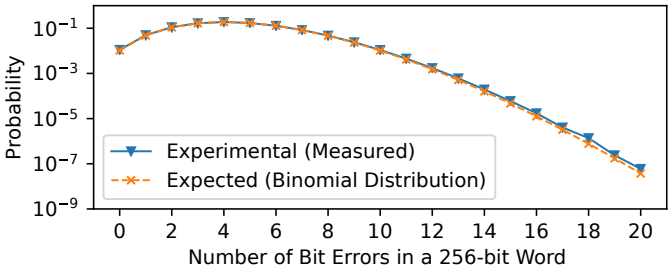

Figure 5: Expected and experimental probabilities of observing an X-bit error in a 256-bit word for a representative DRAM device without on-die ECC at t_{REFW} = 20s and 60°C.

The experimental data is well predicted by the binomial distribution and diverges only at extreme error counts that have few experimental samples. This validates modeling retention errors using a uniform-random distribution for the devices without on-die ECC. We repeat this experiment across all of our devices without on-die ECC for various word sizes, refresh windows, and temperatures, and we find that the uniform-random model holds across all experiments.

7.2. Determining the True-/Anti-Cell Layout

We reverse-engineer the true-/anti-cell layout in the devices with on-die ECC to ensure that we can accurately model the

pre-correction error distribution in simulation (as described in Section 5.1, we only inject errors in cells programmed to the "charged" state). We do this by studying the locations of data-retention errors after disabling refresh for a long time (e.g., >30 minutes), which causes most cells to leak to their discharged state.[8] Figure 6 illustrates the resulting pattern, showing how individual rows comprise entirely true- or anti-cells, and contiguous groups of either 824 or 400 rows alternate throughout a bank. In simulation, we model each DRAM burst to be entirely composed of either true- or anti-cells with a 50% probability. This accurately models sampling an arbitrary burst from the entire memory address space.

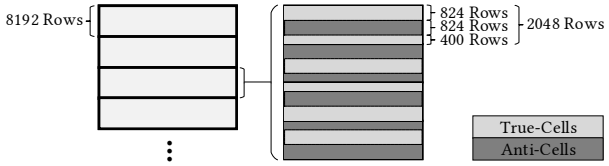

Figure 6: A DRAM bank comprises groups of 824 or 400 rows with alternating true- and anti-cells per group.

Despite the observed true-/anti-cell pattern, we find that a small amount of uniquely randomly-distributed rows in each bank *do not* follow the pattern shown in Figure 6. Instead, these rows alternate true- and anti-cells *every byte* and are often found in clusters of two or more. A histogram of the number of such rows, called *outlier rows*, per bank across all 232 devices with on-die ECC is shown in Figure 7 alongside a best-fit negative-binomial distribution curve. Both the shape of the frequency distribution and the observed clustering are consistent with post-manufacturing repair row remapping techniques [38]. Since these rows have a different true- and anti-cell composition, they add unwanted noise to our reverse-engineering analysis. While we could account for them in our simulations, we simply skip testing these rows in our experimental analysis to avoid unnecessary complexity.

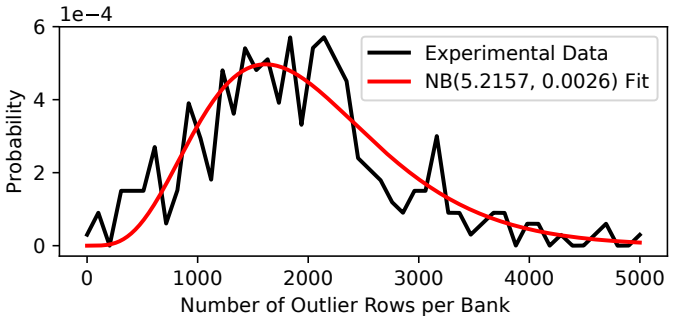

Figure 7: Histogram of the number of rows with outlier true-/anti-cell layouts per bank across all banks of all DRAM devices with on-die ECC (NB: negative-binomial).

7.3. Applying EIN to DRAM with On-Die ECC

We demonstrate applying EIN to the DRAM devices with on-die ECC using the experimental configuration shown in Table 1. The error distribution resulting from a single experiment provides the PMF given by Equation 3, which forms the observations $\mathcal{O}$ in the overall optimization problem (Equation 9).

[8] A small number of cells do not follow the overall pattern due to either *i*) extraordinarily long retention times or *ii*) ECC correction.

Model Parameter	Experiment	Simulation
Word Size	256-bits	
True-/Anti-Cell Layout	50%/50% at word-granularity	
Data Pattern	`RANDOM`	`RANDOM` and `0xFF`
Outlier Rows	Skipped	Ignored
Temperature	70°C	Encompassed in the RBER
t_{REFW}	5 minutes	Encompassed in the RBER

Table 1: Experimental and simulation setup for reverse-engineering the ECC scheme used in the tested devices.

Using a representative device, we perform a single experiment at the conditions shown in Table 1, measuring a *post-correction* BER of 0.041578. Then, configuring EINSim with the parameters listed under "Simulation" in Table 1, we evaluate the full optimization problem of Equation 9 using the grid-search approach described in Section 5.4.

Figure 8 presents the negative log-likelihoods (Equation 8) of the eight highest-likelihood ECC schemes for each of the `0xFF` and `RANDOM` data patterns. Models are sorted in order of increasing likelihood (i.e., *decreasing* negative log-likelihood) from left to right for each data pattern. Bars show black confidence intervals spanning the min/max values when bootstrapping the observed data 10^5 times (described in Section 5.5). The confidence intervals are tight enough to appear as a single line atop each bar. Note that the `0xFF` models have low likelihoods (i.e., higher bars), which agrees with the fact that our experimental data is obtained using a `RANDOM` data pattern.

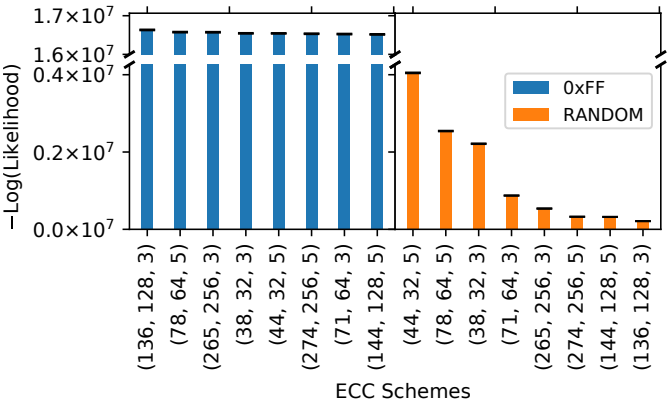

Figure 8: Likelihoods of eight different ECC schemes across two different data patterns, where each likelihood is individually maximized over the model parameter θ.

The smallest (i.e., rightmost) bar represents the *most likely model*, which provides the final reverse-engineered model parameters, including the ECC scheme and the pre-correction error rate. For greater insight into the results, Table 2 describes the five highest-likelihood models in detail.

−Log-Likelihood		f_i	θ	
Value	Bootstrap (Min, Max)	ECC Code	RBER	Data Pattern
2.12e-5	(2.09e-5, 2.15e-5)	(136, 128, 3)	0.038326	RANDOM
3.21e-5	(3.18e-5, 3.24e-5)	(144, 128, 5)	0.039113	RANDOM
3.26e-5	(3.22e-5, 3.29e-5)	(274, 256, 5)	0.039995	RANDOM
5.38e-5	(5.32e-5, 5.43e-5)	(265, 256, 3)	0.039956	RANDOM
8.74e-5	(8.69e-5, 8.79e-5)	(71, 64, 3)	0.038472	RANDOM

Table 2: Details of the five highest-likelihood models (shown in Figure 8) and their raw likelihood values.

The data indicates that a Hamming single-error correction code with (n = 136, k = 128, d = 3) is the most likely ECC

scheme out of all models considered. This result is consistent with several industrial prior works [17, 67, 68, 86].[9] Compared to most of the other codes we consider, (136, 128, 3) code has a relatively low error-correction capability (i.e., 1 bit per 136 codeword bits), which is reasonable for a first-generation on-die ECC mechanism and requires a relatively simple, low-overhead circuit implementation.

The MAP estimate of θ provides the most likely pre-correction error rate (i.e., RBER) and data pattern to explain the observed data. Note that on-die ECC actually *increases* the error rate at these testing conditions, likely due to a high incidence of miscorrections as described in Section 5.3. EIN correctly infers that our experiment uses the `RANDOM` data pattern, which is indicated by the relatively low likelihoods of the models that assume a `0xFF` data pattern.

Figure 9 shows the full PMF of Equation 4 for all sixteen models considered in Figure 8. The maximum-a-posteriori model (dashed) and the experimental data (solid) are shown alongside all other models (dotted). When shown graphically, it is clear that EIN effectively performs a rigorous best-fit analysis over several models to the experimental observations.

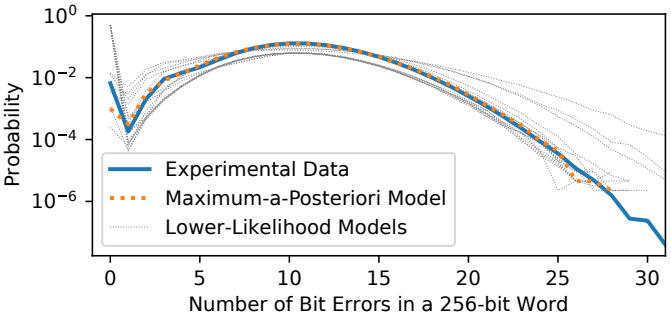

Figure 9: Full PMF for each model considered in Figure 8.

We repeat this analysis across different devices, temperatures, refresh windows, and data patterns and consistently find the (136, 128, 3) ECC code to be the maximum-a-posteriori model. Thus, we conclude that the (136, 128, 3) ECC code is the ECC scheme used in the tested devices.

We draw three key conclusions from this application of EIN. **First**, EIN infers the on-die ECC scheme with *no* visibility into the encoded data or error-detection/-correction information, *without* disabling the ECC mechanism, and *without* tampering with the hardware in any way.

Second, EIN can simultaneously infer several components of θ that might not be known. While we demonstrate a simple inference over only two data patterns in addition to the pre-correction error rate, we could also infer other characteristics (e.g., true-/anti-cell composition, refresh window, temperature). In general, θ is extensible to any model parameter that can be implemented in simulation (i.e., in EINSim).

Third, Figure 9 shows that the maximum-a-posteriori model is a good fit for the empirical data, which supports our assumption that data-retention errors can be modeled as uniformly-random events (Section 4.3) even for devices with on-die ECC.

[9] We obtained this result *without* informing the prior distribution about the existence of prior works. If instead we had done so as mentioned in Section 4.4, (136, 128, 3) [17, 67, 68, 86] and (71, 64, 3) [41, 43] ECC code would have been more overwhelmingly likely.

8. Data-Retention Error Characterization of DRAM Devices with On-Die ECC

Having reverse-engineered the on-die ECC scheme, we characterize data-retention error rates with respect to both t_{REFW} and temperature to demonstrate how EIN enables studying *pre-correction* errors in practice. To our knowledge, this is the *first* work to provide a system-level error characterization study of DRAM devices with on-die ECC in open literature in an effort to understand the pre-correction error characteristics.

8.1. Data-Retention Error Rate vs. Refresh Window

Figure 10 shows the measured data-retention error rates for DRAM devices with and without on-die ECC using different t_{REFW} values at a fixed temperature of 50°C using a `0xFF` data pattern. Each of the five distributions shows the minimum and maximum error rates observed for different groups of devices organized by manufacturer. We also show the *pre-correction* error rates inferred using EIN.

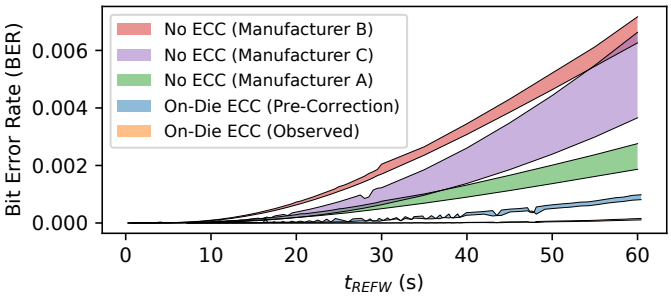

Figure 10: Comparison of data-retention error rates measured using devices with and without on-die ECC, including the inferred pre-correction error rates for devices with on-die ECC.

The data shows that the observed error rates for devices *with* on-die ECC lie far below those of devices *without* on-die ECC. This is consistent with observations from prior works [67,68,86], which find that on-die ECC is a strong enough error-mitigation mechanism to allow for refresh rate reduction. Unfortunately, the *observed* error rates do not provide insight into how the core DRAM technology has changed because it is unclear how much of the error margin improvement is simply a result of ECC.

EIN solves this problem. By inferring the pre-correction error rates, we observe considerable error margin for *even the pre-correction error rates*, implying that on-die ECC may be *unnecessary* at these testing conditions. This may seem surprising at first sight, since error rates are believed to be increasing with technology generation [49, 85, 86, 93]. However, on-die ECC's goal is to combat single-cell errors at *worst-case operating specifications* [86] (i.e., 85°C, t_{REFW} = 32ms [44], worst-case usage characteristics). Unfortunately, our testing infrastructure currently cannot achieve such conditions, and even if it could, the pathological access- and data-patterns depend on the proprietary internal circuit design known only to the manufacturer. Therefore, our observations do not contradict expectations, and we conclude that for devices with on-die ECC: *i*) on-die ECC effectively reduces the observed error rate and *ii*) both pre- and post-correction error rates are considerably lower than those of devices without on-die ECC *at our testing conditions*.

This example demonstrates EIN's strengths: EIN *separates* the effects of a device's particular ECC mechanism from the raw error rates of the DRAM technology and *enables* a meaningful comparison of error characteristics between devices with (or without) different ECC schemes. EIN enables this analysis for *any* error mechanism that EIN is applicable to (Section 5.6).

8.2. Data-Retention Error Rate vs. Temperature

Data-retention error rates are well-known to follow an exponential relationship with respect to temperature [4, 30, 79], and prior works [55, 64, 79] exploit this relationship to extrapolate error rates beyond experimentally feasible testing conditions. We show that on-die ECC distorts this exponential relationship such that observed error rates cannot be reliably extrapolated, and EIN recovers the underlying exponential relationship.

Figure 11 shows the exponential relationship for a single representative device with on-die ECC at a fixed refresh window of 10s on a semilog scale. Measurements (orange, ×) are taken between the temperature limits of our infrastructure (55°C - 70°C, illustrated with a grey background), and the inferred pre-correction error rates (blue, +) and the hypothetical error rates if the on-die ECC scheme were a stronger double-error correction (144, 128, 5) code (green, ∗) are shown. We show exponential fits to data within the measurable region for all three curves. Outside of the measurable region (i.e., <55°C and >70°C), we use EINSim to extrapolate the two post-correction curves beyond the measurable region (dashed) based on the exponential fit for the pre-correction curve.

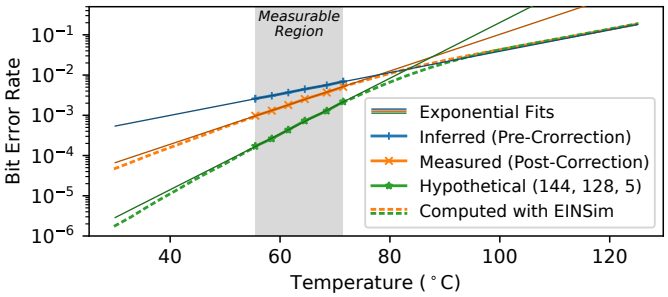

Figure 11: Data-retention error rates of a single representative device with t_{REFW} = 10s across different temperatures, showing error rates: i) measured (post-correction), ii) inferred (pre-correction), and iii) hypothetical post-correction assuming a (144, 128, 5) ECC scheme.

While all three curves *appear* to fit an exponential curve within the measurable temperature range, this is a misleading artifact of sampling only a small fraction of the overall error distribution. Across the full range, only the pre-correction curve follows the exponential relationship: both post-correction curves diverge from the exponential fit on both sides of the measurable region and follow an *ECC-specific* shape. This means that post-correction error rates cannot be directly fitted to an exponential curve, and extrapolating along the known exponential relationship of the data-retention error mechanism requires knowing the pre-correction error rates.

This example demonstrates how EIN recovers the statistical characteristics of the pre-correction error rates that on-die ECC obfuscates. In general, EIN enables this for any error mechanism that EIN is applicable to (discussed in Section 4.3), allowing future works to make use of well-studied error characteristics for devices with ECC.

9. Related Work

To our knowledge, no other work provides and experimentally demonstrates a methodology to infer *i*) the ECC scheme and *ii*) pre-correction error characteristics of a DRAM device with on-die ECC, without access to the device's implementation details. We briefly survey and differentiate our work from related works that are categorized based on their goals.

Reverse-Engineering ECC. Prior works provide techniques for reverse-engineering ECC schemes in NAND flash memories [122, 123, 131] and rank-level ECC DRAMs [19]. However, *none* of these works provide a methodology by which to reverse-engineer an ECC scheme *without visibility into the ECC mechanism*. These works rely on observing the encoded data through a side-channel (e.g., cold-boot attacks [19], directly probing the underlying memory [19, 122, 123, 131]), or knowing when an ECC correction occurs (e.g., timing attacks [19], custom drivers [19]).

In contrast, on-die ECC provides *no such visibility* into the error correction mechanism, and our analysis relies purely on measuring the statistical properties of post-correction errors.

On-Die ECC. Prior works examine on-die ECC as an exploitable mechanism for additional system benefits, including refresh rate reduction [67], standby power reduction [68], and reliability improvement [93]. Our work is the first to propose a general methodology for inferring the on-die ECC scheme and pre-correction error rates.

DRAM Error Characterization. Prior works [5, 10–12, 15, 33, 34, 45–47, 50–53, 56, 57, 57, 58, 60, 61, 70–72, 78–81, 85, 92, 95, 98, 106, 115–118, 121, 124–126, 128, 129, 134] study both data-retention and reduced-latency errors in DDR3 and LPDDR4 DRAM devices. To our knowledge, our work is the first to characterize commodity DRAM devices *with on-die ECC*.

10. Conclusion

We develop EIN, the first statistical inference methodology capable of determining the ECC scheme and pre-correction error rates of a DRAM device with on-die ECC. We provide EINSim [1], a flexible open-source simulator that can apply EIN across different DRAM devices and error models. We evaluate EIN with the *first* experimental study of 232 (82) LPDDR4 DRAM devices with (without) on-die ECC. Using EIN, we: *i*) find that the ECC scheme employed in the devices we test is a single-error correction Hamming code with (n = 136, k = 128, d = 3), *ii*) infer *pre-correction* error rates from *post-correction* errors, and *iii*) recover well-known pre-correction error distributions that on-die ECC obfuscates. With this, we demonstrate that EIN enables DRAM error characterization studies for devices with on-die ECC. We believe and hope that future work will use EIN to develop new understanding and mechanisms to tackle the DRAM scaling challenges that lie ahead.

Acknowledgements

We thank our shepherd Vilas Sridharan, anonymous reviewers, and SAFARI group members for their valuable feedback.

References

[1] "EINSim Source Code," https://github.com/CMU-SAFARI/EINSim.

[2] "Terminology and Notations," http://sites.ieee.org/tcrts/education/terminology-and-notation/, 2018.

[3] R. D. Adams, *High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test*. Springer SBM, 2002.

[4] A. Bacchini *et al.*, "Characterization of Data Retention Faults in DRAM Devices," in *DFT*, 2014.

[5] S. Baek *et al.*, "Refresh Now and Then," in *TC*, 2014.

[6] R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Information and control*, 1960.

[7] Y. Cai *et al.*, "Error Characterization, Mitigation, and Recovery In Flash-Memory-Based Solid-State Drives," *Proc. of the IEEE*, 2017.

[8] Y. Cai *et al.*, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.

[9] S. Cha *et al.*, "Defect Analysis and Cost-Effective Resilience Architecture for Future DRAM Devices," in *HPCA*, 2017.

[10] K. Chandrasekar *et al.*, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in *DATE*, 2014.

[11] K. K. Chang, "Understanding and Improving Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2017.

[12] K. K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.

[13] K. K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.

[14] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[15] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.

[16] M. H. Cho *et al.*, "A Novel Method to Characterize DRAM Process Variation by the Analyzing Stochastic Properties of Retention Time Distribution," in *EDTM*, 2017.

[17] K. C. Chun *et al.*, "A 16Gb LPDDR4X SDRAM with an NBTI-Tolerant Circuit Solution, an SWD PMOS GIDL Reduction Technique, an Adaptive Gear-Down Scheme and a Metastable-Free DQS Aligner In a 10nm Class DRAM Process," in *ISSCC*, 2018.

[18] G. C. Clark Jr and J. B. Cain, *Error-Correction Coding for Digital Communications*. Springer SBM, 2013.

[19] L. Cojocar *et al.*, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *S&P*, 2019.

[20] D. J. Costello and S. Lin, "Error Control Coding: Fundamentals and Applications," 1982.

[21] F. Courbon *et al.*, "Reverse Engineering Flash EEPROM Memories Using Scanning Electron Microscopy," in *CARDIS*, 2016.

[22] T. M. Cover *et al.*, "Elements of Information Theory," *SIAM Review*, 1994.

[23] S. Desai, "Process Variation Aware DRAM (Dynamic Random Access Memory) Design Using Block-Based Adaptive Body Biasing Algorithm," Ph.D. dissertation, Utah State University, 2012.

[24] A. Ditali *et al.*, "X-Ray Radiation Effect in DRAM Retention Time," *T-DMR*, 2007.

[25] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," in *Breakthroughs in Statistics*, 1992.

[26] A. Fukami *et al.*, "Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices," in *Digital Investigation*, 2017.

[27] S. Ghose *et al.*, "What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study," *SIGMETRICS*, 2018.

[28] B. Gu *et al.*, "Challenges and Future Directions of Laser Fuse Processing in Memory Repair," *Proc. Semicon China*, 2003.

[29] T. Hamamoto *et al.*, "Well Concentration: A Novel Scaling Limitation Factor Derived From DRAM Retention Time and Its Modeling," in *IEDM*, 1995.

[30] T. Hamamoto *et al.*, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," in *TED*, 1998.

[31] R. W. Hamming, "Error Detecting and Error Correcting Codes," in *Bell Labs Technical Journal*, 1950.

[32] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.

[33] H. Hassan *et al.*, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.

[34] W. Henkels *et al.*, "A 4-Mb Low-Temperature DRAM," *JSSC*, 1991.

[35] A. Hiraiwa *et al.*, "Statistical Modeling of Dynamic Random Access Memory Data Retention Characteristics," *JAP*, 1996.

[36] A. Hocquenghem, "Codes Correcteurs D'erreurs," *Chiffres*, 1959.

[37] S. Hong, "Memory Technology Trend and Future Challenges," in *IEDM*, 2010.

[38] M. Horiguchi and K. Itoh, *Nanoscale Memory Repair*. Springer SBM, 2011.

[39] W. C. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2010.

[40] A. A. Hwang *et al.*, "Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design," in *SIGPLAN Notices*, 2012.

[41] Intelligent Memory, "I'M ECC DRAM with Integrated Error Correcting Code," 2016, product Brief.

[42] T. Ishigaki *et al.*, "A Multi-Level-Cell Spin-Transfer Torque Memory with Series-Stacked Magnetotunnel Junctions," in *VLSI*, 2010.

[43] *128Mx8, 64Mx16 1Gb DDR3 SDRAM with ECC*, ISSI, 2018, rev. B1.

[44] JEDEC, "Low Power Double Data Rate 4 (LPDDR4) SDRAM Specification," *JEDEC Standard JESD209–4B*, 2014.

[45] M. Jung *et al.*, "Reverse Engineering of DRAMs: Row Hammer with Crosshair," in *MEMSYS*, 2016.

[46] M. Jung *et al.*, "Optimized Active and Power-Down Mode Refresh Control in 3D-DRAMs," in *VLSI-SoC*, 2014.

[47] M. Jung *et al.*, "Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs," in *MEMSYS*, 2015.

[48] H. L. Kalter *et al.*, "A 50-Ns 16-Mb DRAM with a 10-Ns Data Rate and On-Chip ECC," *JSSC*, 1990.

[49] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.

[50] C. Keller *et al.*, "Dynamic Memory-Based Physically Unclonable Function for the Generation of Unique Identifiers and True Random Numbers," in *ISCAS*, 2014.

[51] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.

[52] S. Khan *et al.*, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.

[53] S. Khan *et al.*, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," in *IEEE CAL*, 2016.

[54] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.

[55] I. Kim *et al.*, "High Performance PRAM Cell Scalable to Sub-20nm Technology with Below 4F2 Cell Size, Extendable to DRAM Applications," in *VLSIT*, 2010.

[56] J. S. Kim *et al.*, "Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines," in *ICCD*, 2018.

[57] J. S. Kim *et al.*, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.

[58] J. S. Kim *et al.*, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers With Low Latency And High Throughput," in *HPCA*, 2019.

[59] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," in *EDL*, 2009.

[60] Y. Kim, "Architectural Techniques to Enhance DRAM Scaling," Ph.D. dissertation, Carnegie Mellon University, 2015.

[61] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.

[62] Y. Kim *et al.*, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.

[63] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," in *IEEE CAL*, 2016.

[64] W. Kong *et al.*, "Analysis of Retention Time Distribution of Embedded DRAM-A New Method to Characterize Across-Chip Threshold Voltage Variation," in *ITC*, 2008.

[65] K. Kraft *et al.*, "Improving the Error Behavior of DRAM by Exploiting its Z-Channel Property," in *DATE*, 2018.

[66] E. Kültürsay *et al.*, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," in *ISPASS*, 2013.

[67] N. Kwak *et al.*, "A 4.8 Gb/s/pin 2Gb LPDDR4 SDRAM with Sub-100μA Self-Refresh Current for IoT Applications," in *ISSCC*, 2017.

[68] H.-J. Kwon *et al.*, "An Extremely Low-Standby-Power 3.733 Gb/s/pin 2Gb LPDDR4 SDRAM for Wearable Devices," in *ISSCC*, 2017.

[69] B. C. Lee *et al.*, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.

[70] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.

[71] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon University, 2016.

[72] D. Lee *et al.*, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.

[73] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.

[74] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.

[75] M. J. Lee and K. W. Park, "A Mechanism for Dependence of Refresh Time on Data Pattern in DRAM," in *EDL*, 2010.

[76] Y. Li *et al.*, "DRAM Yield Analysis and Optimization by a Statistical Design Approach," in *CSI*, 2011.

[77] U. Lieneweg *et al.*, "Assesment of DRAM Reliability from Retention Time Measurements," *Flight Readiness Technol. Assessment NASA EEE Parts Prog.*, 1998.

[78] C. H. Lin *et al.*, "SECRET: Selective Error Correction for Refresh Energy Reduction in DRAMs," in *ICCD*, 2012.

[79] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.

[80] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.

[81] W. Liu *et al.*, "A Trustworthy Key Generation Prototype Based on DDR3 PUF for Wireless Sensor Networks," in *Sensors*, 2014.

[82] Y. Luo *et al.*, "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory," in *DSN*, 2014.

[83] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Elsevier, 1977.

[84] J. A. Mandelman *et al.*, "Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM)," in *IBM JRD*, 2002.

[85] J. Meza *et al.*, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *DSN*, 2015.

[86] Micron Technology Inc., "ECC Brings Reliability and Power Efficiency to Mobile Devices," Micron Technology Inc., Tech. Rep., 2017.

[87] Y. Mori *et al.*, "A New Method for Predicting Distribution of DRAM Retention Time," in *IRPS*, 2001.

[88] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.

[89] O. Mutlu, "The RowHammer Problem and Other Issues we may Face as Memory Becomes Denser," in *DATE*, 2017.

[90] O. Mutlu and J. Kim, "RowHammer: A Retrospective," in *TCAD*, 2019.

[91] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," in *SUPERFRI*, 2014.

[92] P. J. Nair *et al.*, "ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates," in *ISCA*, 2013.

[93] P. J. Nair *et al.*, "XED: Exposing On-Die Error Detection Information for Strong Memory Reliability," in *ISCA*, 2016.

[94] T.-Y. Oh *et al.*, "A 3.2Gbps/pin 8Gb 1.0V LPDDR4 SDRAM with Integrated ECC Engine for Sub-1V DRAM Core Operation," in *ISSCC*, 2014.

[95] T. Ohsawa *et al.*, "Optimizing the DRAM Refresh Count for Merged DRAM/logic LSIs," in *ISLPED*, 1998.

[96] K. Park *et al.*, "Experiments and Root Cause Analysis for Active-Precharge Hammering Fault In DDR3 SDRAM Under 3× Nm Technology," *Microelectronics Reliability*, 2016.

[97] K. Park *et al.*, "Statistical Distributions of Row-Hammering Induced Failures in DDR3 Components," *Microelectronics Reliability*, 2016.

[98] M. Patel *et al.*, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.

[99] M. K. Qureshi *et al.*, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.

[100] M. K. Qureshi *et al.*, "Scalable High Performance Main Memory System Using Phase-change Memory Technology," in *ISCA*, 2009.

[101] A. Rahmati *et al.*, "Probable Cause: The Deanonymizing Effects of Approximate DRAM," in *ISCA*, 2016.

[102] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[103] R. M. Roth, *Introductin to Coding Theory*. Cambridge University Press, 2006.

[104] K. Saino *et al.*, "Impact of Gate-Induced Drain Leakage Current on the Tail Distribution of DRAM Data Retention Time," in *IEDM*, 2000.

[105] A. Schaller *et al.*, "Intrinsic Rowhammer PUFs: Leveraging the Rowhammer Effect for Improved Security," in *HOST*, 2017.

[106] B. Schroeder *et al.*, "DRAM Errors in the Wild: a Large-Scale Field Study," in *SIGMETRICS*, 2009.

[107] N. H. Seong *et al.*, "Tri-Level-Cell Phase Change Memory: Toward an Efficient and Reliable Memory System," in *ISCA*, 2013.

[108] V. Seshadri, "Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2016.

[109] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[110] V. Seshadri *et al.*, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.

[111] S. M. Seyedzadeh *et al.*, "Mitigating Bitline Crosstalk Noise in DRAM Memories," in *ISMS*, 2017.

[112] C. G. Shirley and W. R. Daasch, "Copula Models of Correlation: A DRAM Case Study," in *TC*, 2014.

[113] Y. H. Son *et al.*, "CiDRA: A cache-Inspired DRAM resilience architecture," in *HPCA*, 2015.

[114] V. Sridharan *et al.*, "Memory Errors in Modern Systems: The Good, the Bad, and the Ugly," in *ASPLOS*, 2015.

[115] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," in *SC*, 2012.

[116] S. Sutar *et al.*, "D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication and Random Number Generation," *TECS*, 2018.

[117] B. Talukder *et al.*, "Exploiting DRAM Latency Variations for Generating True Random Numbers," *arXiv preprint arXiv:1808.02068*, 2018.

[118] B. Talukder *et al.*, "LDPUF: Exploiting DRAM Latency Variations to Generate Robust Device Signatures," *arXiv preprint arXiv:1808.02584*, 2018.

[119] Q. Tang *et al.*, "A DRAM Based Physical Unclonable Function Capable of Generating ¿10^{32} Challenge Response Pairs per 1Kbit Array for Secure Chip Authentication," in *CICC*, 2017.

[120] F. Tehranipoor *et al.*, "Investigation of DRAM PUFs Reliability Under Device Accelerated Aging Effects," in *ISCAS*, 2017.

[121] F. Tehranipoor *et al.*, "Robust Hardware True Random Number Generators using DRAM Remanence Effects," in *HOST*, 2016.

[122] J. P. van Zandwijk, "A Mathematical Approach to NAND Flash-Memory Descrambling and Decoding," *Digital Investigation*, 2015.

[123] J. P. van Zandwijk, "Bit-Errors as a Source of Forensic Information in NAND-Flash Memory," *Digital Investigation*, 2017.

[124] R. K. Venkatesan *et al.*, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *HPCA*, 2006.

[125] F. Wang *et al.*, "DRAM Retention at Cryogenic Temperatures," in *IMW*, 2018.

[126] J. Wang *et al.*, "ProactiveDRAM: A DRAM-Initiated Retention Management Scheme," in *ICCD*, 2014.

[127] A. Weber *et al.*, "Data Retention Analysis on Individual Cells of 256Mb DRAM in 110nm Technology," in *ESSDERC*, 2005.

[128] C. Weis *et al.*, "Retention Time Measurements and Modelling of Bit Error Rates of Wide I/O DRAM in MPSoCs," in *DATE*, 2015.

[129] C. Weis *et al.*, "Thermal Aspects and High-Level Explorations of 3D Stacked DRAMs," in *ISVLSI*, 2015.

[130] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*. Prentice Hall Englewood Cliffs, 1995.

[131] J. Wise, "Reverse Engineering a NAND Flash Device Management Algorithm," https://joshuawise.com/projects/ndfrecovery#ecc_recovery, 2014.

[132] H.-S. P. Wong *et al.*, "Metal–Oxide RRAM," *Proc. IEEE*, 2012.

[133] H.-S. P. Wong *et al.*, "Phase Change Memory," *Proceedings of the IEEE*, 2010.

[134] W. Xiong *et al.*, "Run-Time Accessible DRAM PUFs in Commodity Devices," in *CHES*, 2016.

[135] C. Zhang *et al.*, "Hi-Fi Playback: Tolerating Position Errors in Shift Operations of Racetrack Memory," in *ISCA*, 2015.

[136] T. Zhang *et al.*, "Half-DRAM: A High-Bandwidth and Low-Power DRAM Architecture from the Rethinking of Fine-Grained Activation," in *ISCA*, 2014.

[137] X. Zhang *et al.*, "Exploiting DRAM Restore Time Variations In Deep Sub-Micron Scaling," in *DATE*, 2015.

[138] Y. Zhang *et al.*, "Multi-Level Cell STT-RAM: Is it Realistic or Just a Dream?" in *ICCAD*, 2012.