creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학박사학위논문

데이터 집약적 응용을 위한 인플래시 프로세싱 연구

In-Flash Processing for Data-Intensive Applications

2026년 2월

서울대학교 대학원
컴퓨터공학부
이재용

데이터 집약적 응용을 위한
인플래시 프로세싱 연구

In-Flash Processing
for Data-Intensive Applications

지도교수 김 지 홍

이 논문을 공학박사 학위논문으로 제출함
2025년 12월

서울대학교 대학원
컴퓨터공학부
이재용

이재용의 공학박사 학위논문을 인준함
2026년 1월

위 원 장 이 재 욱 (인)

부위원장 김 지 홍 (인)

위 원 안 정 호 (인)

위 원 김 남 승 (인)

위 원 김 명 석 (인)

Abstract

NAND flash-based Solid-State Drives (SSDs) have become the predominant storage medium for modern data-intensive applications. Compared to traditional Hard Disk Drives (HDDs), SSDs offer various advantages, including shorter I/O latency, compact form factor, and lower energy consumption. Moreover, advances in multi-level cell technology and 3D wordline stacking have significantly increased SSD capacity, enabling a single device to store hundreds of terabytes of data. With these benefits and remarkable capacity enhancements, SSDs have emerged as the de facto standard for modern data-intensive workloads.

While SSDs provide significantly higher performance than HDDs, their limited bandwidth still makes it challenging to meet the demanding performance requirements of data-intensive applications. Modern data-intensive workloads demand extremely high data processing capability over massive datasets. To satisfy these requirements, modern computing systems employ high-performance compute units (e.g., GPUs) capable of delivering hundreds of tera–floating-point operations per second. Unfortunately, the bandwidth of SSDs, which physically store the data to be processed, remains limited to about 4–15 GB/s. This severely constrained SSD bandwidth introduces a substantial performance bottleneck that cannot be addressed merely by increasing compute performance.

To overcome the limited bandwidth of SSDs, in-flash processing (IFP), which performs computation directly within flash chips inside the SSD,

has been actively investigated. Because the aggregated internal bandwidth across multiple flash chips within an SSD is higher than the external PCIe interface bandwidth, IFP can effectively increase the available bandwidth of SSDs. Leveraging this advantage, several studies have proposed IFP-based acceleration techniques for various data-intensive applications.

IFP has the potential to deliver substantial performance benefits, but several key challenges must be addressed before it can be adopted in practical systems. First, achieving high bandwidth requires IFP to operate all flash chips concurrently, which exposes the SSD to significant read disturbance that severely degrades both its lifetime and performance. Second, the capacity-oriented design trend of modern NAND flash memory has substantially reduced per-chip read bandwidth, directly undermining the performance benefits that IFP aims to deliver. Third, flash chips are manufactured under extremely tight fabrication resource constraints, making the integration of additional IFP logic components both challenging and costly. In particular, ensuring reliable IFP operation necessitates an on-chip ECC decoder, which incurs substantial area and power overhead.

In this dissertation, we propose two novel schemes that effectively address the key challenges hindering the practical adoption of IFP. First, we introduce DEAR, a dynamic read-refresh scheme that mitigates the impact of the read disturbance by performing refreshes in a fine-grained manner. Conventional SSD firmware ensures reliability against read disturbance by simply triggering a refresh once the number of reads to a block exceeds a predefined threshold. However, due to the substantial process variation in modern 3D NAND flash memory, such coarse-grained refresh policies cause

excessive and unnecessary data migrations. To eliminate this inefficiency, DEAR performs read-refresh operations based on actual error rates rather than read counts, and operates at the wordline granularity instead of the block level. By selectively refreshing only the highly disturbed wordlines, DEAR drastically reduces the data migrations during refreshes, thereby significantly improving both SSD lifetime and performance under read disturbance.

Second, we present AiF, an in-flash processing framework that effectively improves bandwidth and implementation overheads of IFP. To overcome the limited performance benefits of IFP caused by the diminishing read bandwidth of flash chips, AiF employs a charge-recycling read technique that dramatically increases read bandwidth by reusing the voltages applied in previous read operations for subsequent reads. In addition, to address the area and power overhead of on-chip ECC decoders required to correct the high error rate of NAND flash memory, AiF adopts biased error encoding, which intentionally introduces non-uniform error characteristics across pages to create highly reliable pages. By selectively storing IFP data (e.g., LLM weights) only in these reliable pages, AiF ensures the reliability of IFP using lightweight ECC. Combining these two techniques, AiF achieves up to 4x higher read bandwidth than conventional IFP while significantly reducing implementation overhead.

To evaluate the effectiveness of the proposed schemes, we conduct extensive experiments using real-world workloads and LLMs on SSD emulation platforms. The evaluation results show that DEAR significantly reduces data copying caused by read-refresh operations, thereby substantially im-

proving both SSD lifetime and performance for read-intensive workloads. Furthermore, the IFP-specialized read and reliability techniques of AiF enable a dramatic increase in read bandwidth with marginal implementation overhead, leading to a significant improvement in LLM inference throughput.

Keywords: In-Flash Processing, NAND Flash Memory, Storage System, LLM Inference, Performance Optimization, Reliability Optimization

Student Number: 2020-20678

Contents

I. Introduction **1**

1.1 Motivation 1

1.2 Dissertation Goals 3

1.3 Contributions 5

1.4 Dissertation Structure 7

II. Background **10**

2.1 Overview of 3D NAND Flash Memory 10

2.1.1 Organization 10

2.1.2 3D Manufacturing Process 11

2.1.3 NAND Flash Operation 12

2.1.4 Multi-level Cell Technology 14

2.2 Read-Disturbance Problem 15

2.2.1 Read-Disturbance Mechanism 15

2.2.2 Read Disturbances in 3D Flash Memory 16

2.2.3 State-Dependent Read-Disturbance Errors 17

2.2.4 Reliability Management: Read Refresh 18

2.3 Related Work 19

2.3.1 Read-Disturbance Management 19

2.3.2 In-Storage Processing 20

2.3.3 In-Flash Processing 21

III. Key Challenges in In-Flash Processing 22
3.1 Diminishing Read Bandwidth 22
3.1.1 Capacity Scaling of NAND Flash Memory 22
3.1.2 Limitations of IFP for Read-Intensive Workloads . . 25
3.2 High Read Disturbance 26
3.2.1 Read Disturbance in Modern SSDs 26
3.2.2 Read Disturbance in IFP-enabled SSDs 31
3.3 High Implementation Overheads 34
3.3.1 Root Causes of High Implementation Overheads: PPA Inefficiency and ECC Complexity 34
3.3.2 Quantitative Evaluation of On-Chip ECC Overheads 35

IV. Mitigating Read Disturbance via Dynamic Error-Aware Refresh 38
4.1 Limitations of the Existing Solutions 38
4.2 Overview of DEAR 40
4.3 Design of Swift Error Estimator 45
4.3.1 Intra-Layer Similarity for SAM 46
4.3.2 Correlation of Fail-Bit Count for FEM 47
4.3.3 Monitoring Interval Analysis for BPM 49
4.4 DEARSSD: System Integration 51
4.5 Evaluation 55
4.5.1 Evaluation Methodology 55
4.5.2 RR-Induced Writes 57
4.5.3 Read Latency 59

4.5.4 SSD Lifetime 61
4.5.5 Monitoring Overhead 63

V. Specializing Flash Operations for Efficient In-Flash Processing 64

5.1 AiF: Accelerator-in-Flash 65
5.1.1 Overview 65
5.1.2 Charge-Recycling Read 67
5.1.3 Bias-Error Encoding 72
5.1.4 Design of AiFChip 77
5.2 Integrating AiF into On-Device LLM Inference 80
5.2.1 LLM Inference Workflow with AiFSSD 80
5.2.2 System Supports for AiFSSD Integration 82
5.3 System-Level Evaluation 84
5.3.1 Evaluation Setup 84
5.3.2 Throughput 88
5.3.3 Energy Efficiency 90
5.3.4 Scalability 91
5.3.5 Overhead Analysis 92
5.3.6 RR-Induced Writes 93

VI. Conclusions 96

6.1 Summary 96
6.2 Future Work 98
6.2.1 Reliability-Aware Data Placement 98
6.2.2 Accelerating Vector ANNS Using In-Flash Processing 98

6.2.3 Composable In-Storage Processing Framework . . . 100

Appendix **104**

List of Figures

Figure 1. An organizational overview of a 3D NAND flash-based SSD. 11

Figure 2. A cross-sectional view of a flash block. 12

Figure 3. A read sequence of NAND flash memory. 13

Figure 4. V_{TH} distributions of NAND flash memory. 14

Figure 5. A V_{TH} distribution after read-disturbance. 16

Figure 6. A breakdown of per-state errors on 3D TLC flash. . . . 17

Figure 7. Comparison of the cell-array organization in (a) earlier-generation and (b) modern high-density NAND flash memory. 23

Figure 8. Comparison of (a) flash-chip capacity and (b) read bandwidth between Chip A and Chip B. 24

Figure 9. Internal bandwidth of SSDs. 26

Figure 10. A comparison of RR-induced writes in four SSDs. . . . 28

Figure 11. A comparison of read latency in eight SSDs. 30

Figure 12. Changes in C_{max} values in different flash memories. . . 31

Figure 13. A comparison of read latency under different read intensities. 33

Figure 14. Impact of RBER on inference accuracy. 36

Figure 15. PPA impact of ECC decoder. 37

Figure 16. Per-WL C_{max}^{WL} distributions over different P/E cycles. . 39

Figure 17. An illustrative example of the DEAR scheme. 41

Figure 18. (a) Error similarity within a 3D block. Comparison of (b) straightforward error monitoring and (c) SAM. . . . 42
Figure 19. (a) Correlation between C_{FBC} and errors. Comparison of (b) ECC-based error monitoring and (c) FEM. . . . 43
Figure 20. An illustrative example of BPM when the monitoring interval C_{INT} is set to 100 reads. 44
Figure 21. Intra-layer C_{max}^{WL} similarity. 47
Figure 22. Correlation between C_{FBC} and errors. 48
Figure 23. Determination of monitoring interval C_{INT}. 50
Figure 24. C_{FBC} measurement. 52
Figure 25. Interleaved monitoring. 53
Figure 26. RR invocation flow. 54
Figure 27. Comparisons on the number of internal writes at different P/E cycles: 0.5K, 1.0K and 2.0K. 58
Figure 28. The 99th read tail latency at different P/E cycles: 0.5K, 1.0K and 2.0K. 60
Figure 29. The 99.9th read tail latency at different P/E cycles: 0.5K, 1.0K and 2.0K. 61
Figure 30. A comparison of WAF improvement. 62
Figure 31. Monitoring overhead analysis. 63
Figure 32. A structural overview of AiF. 65
Figure 33. A comparison of (a) the baseline SSD offloading scheme and (b) AiF. 66
Figure 34. An operational overview of cr-read. 68

Figure 35. A comparison of (a) conventional read sequence and (b) cr-read. 69
Figure 36. Evaluation results of cr-read. 71
Figure 37. Characteristics of modern SSDs: (a) Variations of internal bandwidth over varying tR and (b) Changes in bit errors over increasing P/E cycles. 73
Figure 38. A comparison of (a) conventional V_{TH} state encoding and (b) proposed V_{TH} state encoding. 75
Figure 39. Impact of V_{TH} state encoding on page error rates. . . . 76
Figure 40. On-chip computation flow. 78
Figure 41. Optimizing LLM inference performance through parallel host-AiFSSD execution: (a) computational flow of LLM inference, (b) a sequential execution flow, and (c) a parallel execution flow. 81
Figure 42. Throughput comparison of In-Memory, Memory+SSD, AiF- -, and AiF. 89
Figure 43. (a) Comparisons of energy efficiency. (b) Throughput scalability in AiF. 91
Figure 44. Overhead analysis. 92
Figure 45. Comparison of RR-induced internal writes for AiF and AiF+ across different P/E cycles (0.5K, 1K, and 2K) over 5000 generated tokens for LLaMA2-13B and Falcon-40B. 94

List of Tables

Table 1. Detailed specifications of 11 SSDs for evaluation. 27

Table 2. Key parameters of a target SSD. 56

Table 3. Key I/O characteristics of evaluated workloads. 57

Table 4. Breakdown of area and power. 79

Table 5. Evaluated SSD configurations. 87

Table 6. Evaluated LLM configurations. 88

Chapter 1

Introduction

1.1 Motivation

To meet the ever-growing performance and capacity demands of modern data-intensive applications, in-flash processing (IFP) has been actively investigated in recent years [1, 2, 3, 4, 5, 6]. As a new paradigm of computational SSDs, IFP executes application-level computations directly inside flash chips, rather than transferring data to external compute units.

By enabling computation within flash chips, IFP offers several key advantages over previous computational SSD solutions such as in-storage processing (ISP) [7, 8, 9, 10, 11, 12], as well as DRAM-based processing-in-memory (PIM) solutions. First, IFP can fully exploit the internal bandwidth of SSDs. Previous ISP solutions that perform computation at the SSD controller must first transfer data from flash chips through flash channels, making their performance inherently constrained by the limited bandwidth of these channels. In contrast, IFP performs computation directly inside flash chips, where the data to be processed is stored, and transfers only the final results. This eliminates the flash-channel bottleneck and enables full utilization of the aggregated bandwidth across all flash chips. Second, because IFP is built on NAND flash memory, it can provide more than 50x higher storage capacity than DRAM-based PIM solutions. These combined bene-

fits in performance and capacity make IFP a promising solution for modern data-intensive applications that require high I/O bandwidth over massive datasets.

While IFP has demonstrated its effectiveness across a wide range of data-intensive applications, several fundamental challenges remain insufficiently addressed in prior work. As the capacity of modern NAND flash memory continues to scale rapidly, the reliability and performance of flash chips degrade significantly, posing serious challenges to IFP, which performs computation directly within flash chips. First, IFP increasingly suffers from severe read disturbance. As flash chip density increases, read-disturbance effects, where errors accumulate with each read operation, are significantly exacerbated. Since IFP operates under read-intensive workloads and typically activates all flash chips concurrently to maximize internal bandwidth, it exposes them to extreme read-disturbance stress. This substantially increases the frequency of read-refresh operations, an internal SSD maintenance task required to ensure reliability, thereby significantly degrading both the lifetime and performance of IFP-enabled SSDs. Second, capacity-oriented flash memory designs, such as multi-level cell technology and 3D wordline stacking, have drastically reduced flash read bandwidth. This bandwidth degradation fundamentally limits the performance benefits that IFP can provide. Finally, flash chips are manufactured under extremely tight area and power constraints, making the integration of additional IFP logic challenging. For instance, IFP often requires the integration of on-chip ECC decoders to ensure reliability, yet the high complexity of ECC decoder incurs substantial area and power overhead.

1.2 Dissertation Goals

Since the reliability and performance issues of IFP stem from multiple underlying factors, it is difficult to address all of them with a single, unified solution. Therefore, effectively overcoming the diverse challenges of IFP requires the development of multiple complementary techniques, each targeting a specific issue, and their careful integration. This holistic approach enables IFP to accelerate a wide range of data-intensive applications with substantial performance gains while incurring minimal additional overhead.

In this dissertation, we propose two novel schemes that effectively address the reliability and performance challenges of IFP that have not been adequately explored in existing solutions. In particular, our primary goal is to identify and exploit new optimization opportunities, ranging from low-level device characteristics to high-level application requirements, that remain unleveraged by prior work. Based on this holistic perspective, we develop software-, microarchitecture-, and circuit-level optimization techniques that comprehensively improve the reliability and performance of IFP-enabled SSDs from multiple dimensions.

First, we present a novel read-refresh scheme that effectively mitigates the lifetime and performance overhead caused by read disturbance under read-intensive workloads. Despite the fact that modern SSDs increasingly suffer from severe read disturbance due to larger block sizes and shrinking error margins, conventional read-refresh schemes still rely on a simplistic mechanism that ensures reliability by triggering refresh operations once a block's read count reaches a predefined threshold. However, we ob-

serve that read-disturbance characteristics in modern 3D NAND flash memory vary significantly across wordlines, making existing block-level, static read-count-based refresh schemes highly inefficient. To address this inefficiency, our proposal selectively refreshes individual wordlines based on their actual error rates, rather than relying on block-level read counts. By enabling refresh operations only for highly disturbed wordlines, our scheme significantly reduces the amount of data copying compared to conventional read-refresh approaches, particularly under read-intensive workloads.

Second, we present a novel IFP-specialized SSD architecture that effectively addresses the performance and reliability challenges of in-flash processing. We observe that, due to the capacity-oriented design of modern 3D NAND flash memory, both the performance and reliability of flash chips have degraded significantly, making it difficult to satisfy the demanding performance and reliability requirements of LLM inference even when employing IFP. To address this challenge, our proposal first significantly improves flash read bandwidth by specializing the flash-chip read sequence, originally optimized for general-purpose I/O, to match the access patterns of IFP workloads. In addition, we introduce a new data-encoding technique that intentionally creates highly reliable pages. By selectively placing IFP data only in these reliable pages, our design ensures error correction using lightweight on-die ECC decoders with minimal overhead. By integrating these two techniques, our proposal significantly accelerates on-device LLM inference with low implementation overhead. Moreover, the proposed IFP-specialized techniques are broadly applicable beyond LLM inference and can benefit a wide range of data-intensive workloads.

1.3 Contributions

In this dissertation, we introduce several reliability and performance optimization techniques that address the fundamental challenges hindering the practical adoption of IFP. The contributions of our work can be summarized as follows:

- We propose Dynamic Error-Aware Refresh (DEAR), a novel read-disturbance management scheme that minimizes unnecessary data copying during read-refresh operations, thereby significantly reducing the lifetime and performance overhead of read refresh under read-intensive workloads such as IFP. The key observation behind DEAR is that, due to substantial process variation in modern 3D NAND flash memory, wordlines exhibit highly heterogeneous read-disturbance characteristics. As a result, conventional block-level read-refresh schemes based solely on read counts often trigger refresh operations prematurely. To address this inefficiency, DEAR introduces an accurate and lightweight online wordline error estimator that continuously tracks error accumulation at the wordline granularity. When the estimated error of a wordline approaches the ECC correction capability, DEAR selectively refreshes only that wordline. By basing refresh decisions on actual error rates rather than read counts, DEAR effectively minimizes unnecessary refresh invocations. Experimental results show that DEAR reduces the number of refresh-induced writes by an average of 64.2% compared to existing read-refresh management schemes. By significantly reducing internal data migration caused by read-refresh

operations, DEAR improves the 99.9th-percentile read latency and SSD lifetime by 44.9% and 36.7%, respectively. Furthermore, thanks to a highly optimized implementation of the wordline error estimator, DEAR incurs negligible error-tracking overhead.

- We propose Accelerator-in-Flash (AiF), a novel IFP architecture that executes the key computation stages of on-device LLM inference directly inside flash chips, thereby significantly improving inference throughput under limited memory capacity. We identify that, due to the low read bandwidth of flash chips, existing IFP schemes fail to meet the demanding bandwidth requirements of LLM inference even when computation is offloaded into flash chip. Moreover, while LLM inference is highly error-sensitive, where even minor bit errors can severely degrade accuracy, flash memory inherently exhibits low reliability. Ensuring correctness therefore requires complex on-die ECC decoders, which incur prohibitive area and power overheads. To address these challenges, AiF introduces two IFP-specialized techniques: charge-recycling read and biased error encoding. Charge-recycling read is based on the key observation that when wordlines within a block are accessed consecutively, the read voltages applied in a previous access can be reused for subsequent reads. Leveraging this property, AiF reads large volumes of consecutively stored model weights with significantly higher effective read bandwidth. Biased error encoding intentionally introduces non-uniform reliability across pages within a block, creating a subset of highly reliable pages. Exploit-

ing this biased reliability, AiF selectively stores model weights only in these reliable pages. As a result, data can be correctly recovered using a highly compact on-die ECC decoder dedicated solely to reliable pages. By combining these two techniques, AiF improves flash read bandwidth by 4x while incurring negligible area and power overheads. Evaluation results using eight real-world LLMs show that AiF achieves up to 14.6× higher throughput than a baseline DRAM+SSD inference scheme. Furthermore, compared to in-memory inference that stores the entire model in device memory, AiF delivers 1.4× higher throughput while significantly reducing memory footprint. The IFP-specialized performance and reliability optimization techniques proposed in AiF are broadly applicable and can be leveraged to accelerate a wide range of data-intensive workloads using IFP.

1.4 Dissertation Structure

This dissertation is composed of six chapters including the introduction and conclusions which are at the first and the last, respectively. The four intermediate chapters are organized as follows:

Chapter 2 provides the background necessary to understand this work. It introduces the fundamentals of NAND flash memory, the overall architecture of SSDs, and read-disturbance management mechanisms in modern SSDs. In addition, this chapter reviews prior work on read-disturbance mitigation techniques as well as existing computational SSD solutions that are closely related to our proposed approaches.

Chapter 3 discusses the key challenges that must be addressed for the practical adoption of IFP. It explains how the aggressive capacity scaling of modern 3D NAND flash memory has significantly degraded both performance and reliability, thereby introducing several critical challenges for IFP, which executes computation directly inside flash chips. This chapter highlights that overcoming these challenges requires new optimization techniques and serves as the foundation for the solutions proposed in the subsequent chapters.

Chapter 4 presents DEAR, an effective read-disturbance management scheme that minimizes unnecessary data migration during read-refresh operations. We first explain the key limitations of existing read-refresh schemes in high-density 3D NAND flash memory, where read-disturbance characteristics are highly heterogeneous across wordlines. To address these limitations, DEAR selectively refreshes only highly disturbed wordlines based on actual error rates rather than read counts. We also introduce a highly optimized wordline error estimator to enable the practical implementation of DEAR. We evaluate the effectiveness of DEAR using real-world workloads, showing large lifetime and performance benefits over existing schemes.

Chapter 5 introduces AiF, a novel IFP scheme that accelerates on-device LLM inference. This chapter first discusses the limitations of existing IFP solutions when applied to workloads with demanding bandwidth and reliability requirements, such as LLM inference. To overcome these challenges, we propose IFP-specialized flash read sequences and data-encoding techniques. We validate the effectiveness of the proposed techniques through industry-standard EDA tools and real device-level characterization. Evalu-

ation results show that AiF significantly improves LLM inference performance, demonstrating that it effectively addresses the limitations of existing IFP solutions.

Chapter 2

Background

2.1 Overview of 3D NAND Flash Memory

2.1.1 Organization

Figure 1 shows the organization of a 3D NAND flash-based SSD. A flash cell is the smallest unit for storing data, holding bit information based on its threshold voltage (V_{TH}) level, which largely depends on the charge in its charge trap layer. It means that different bit data can be encoded by different V_{TH} of the cell. For example, we can assign data value '0' to a higher V_{TH} and '1' to a lower V_{TH}. A set of vertically connected cells forms a NAND string. Each string connects to a bitline (BL), and multiple strings on different BLs form a block. Cells at the same vertical location share a control gate connected to a common wordline (WL), enabling all cells on that WL to operate concurrently. In our example, a 3D flash block consists of *N* horizontal layers (h-layers) stacked along the z-axis. Similarly, the 3D flash block may be described as having four vertical layers (v-layers) in the x-axis where each v-layer consists of *N* stacked WLs separated by four select-line transistors (SSLs). Within each plane, all blocks share the BLs. Each BL is connected to a sense amplifier (not shown in Figure 1) in the peripheral circuitry, which detects the values stored in the cells during read operations.

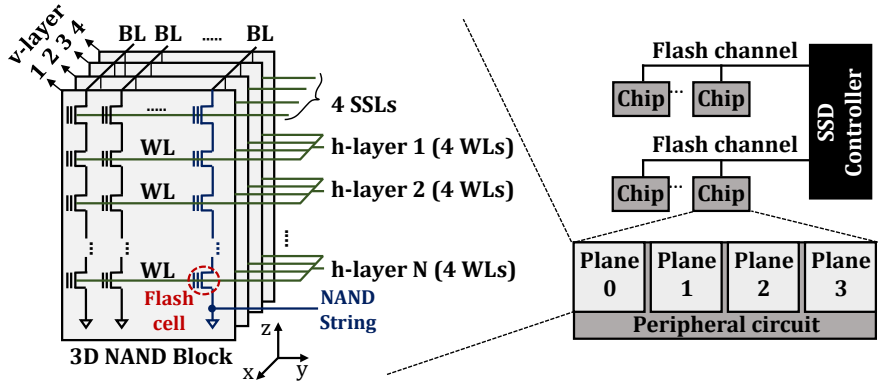


Figure 1: An organizational overview of a 3D NAND flash-based SSD.

Each flash chip, containing multiple planes (typically four), communicates with the controller through a dedicated flash channel.

2.1.2 3D Manufacturing Process

Figure 2 shows a cross-sectional view of a flash block cut in the x-z plane. In 3D flash memory, the stacked flash cells are vertically connected through cylindrical channel holes. The channel holes are formed at the early stage of 3D NAND flash manufacturing by an etching process [13]. While the etching process proceeds from the top h-layer to the bottom h-layer, it causes structural variations to channel holes in different h-layers. For example, the diameter of the channel holes fluctuates considerably across the height of an h-layer. Because of the high aspect ratio of the cylindrical channel hole, the channel hole diameter in the topmost h-layer is wider than that in the bottom h-layer. Moreover, channel holes in some h-layers have an elliptical or rugged shape, unlike the circles in upper h-layers, mainly due to etchant fluid dynamics. These structural variations lead to differences in

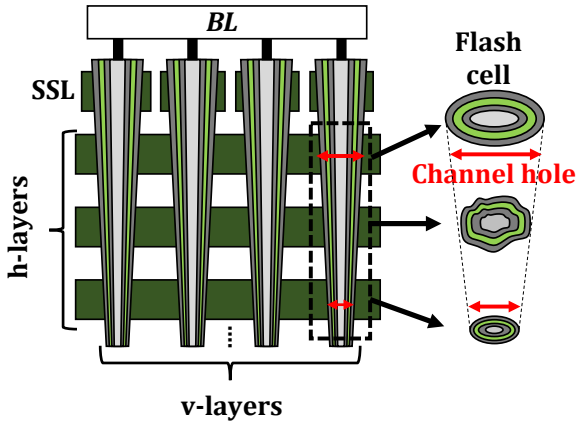


Figure 2: A cross-sectional view of a flash block.

error characteristics of flash cells along the vertical direction.

2.1.3 NAND Flash Operation

There are three basic operations that enable access to flash memory: program, erase, and read. To increase the threshold voltage (V_{TH}) of a flash cell, the program operation injects electrons into the charge trap layer of the flash cell by applying a high voltage (e.g., $>$ 20 V) to the WL. On the contrary, the erase operation applies a high voltage ($>$ 20V) to the substrate (i.e., channel) to remove electrons from the charge trap layer, which decreases the V_{TH} of all flash cells in a block.

During a read operation, NAND flash memory determines a cell's data (i.e., the cell's V_{TH} level) by identifying whether current flows through the corresponding BL. Figure 3 depicts the read mechanism of NAND flash memory that consists of three phases: 1) precharge, 2) evaluation, and 3) discharge.

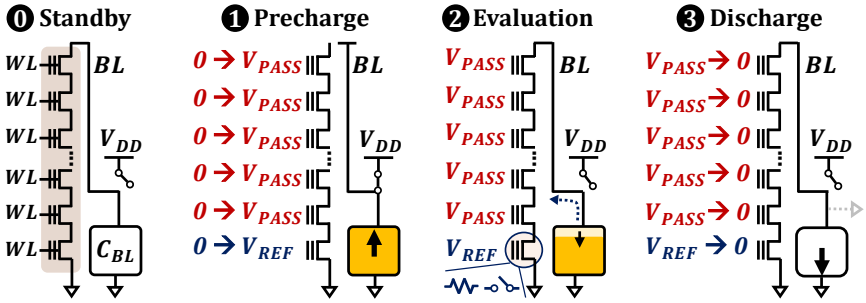


Figure 3: A read sequence of NAND flash memory.

In the precharge phase (❶) in Figure 3), the WLs and BLs are charged to the specific voltages needed to read the values stored on the target WL. A read reference voltage (V_{REF}) is applied to the target WL in the block. Since non-target WLs should not affect the read operation on the target WL, their gate voltages are set to *pass-through voltage* (V_{PASS}, $> 6V$), which is much higher than the highest V_{TH} level of any flash cells in the NAND string (i.e., non-target WLs should behave like pass-transistors.). Simultaneously, the BLs are precharged in preparation for data sensing. During the evaluation phase (❷), the stored values in the target cells are detected. If the (V_{TH}) of the target cell is below V_{REF}, the cell turns on, allowing current to flow. Conversely, if V_{TH} is higher than V_{REF}, the cell remains off, blocking the current. The sense amplifier connected to each BL determines the V_{TH} of the target cell by detecting the current sinking from the BL capacitor (C_{BL}). Due to the high resistance of the BL and NAND string, this sink current is minimal, causing only a slight change in the voltage level of C_{BL}, even when the target cell is on [14]. Finally, in the discharge phase (❸), the WLs and BLs of the target block are discharged to return to standby state

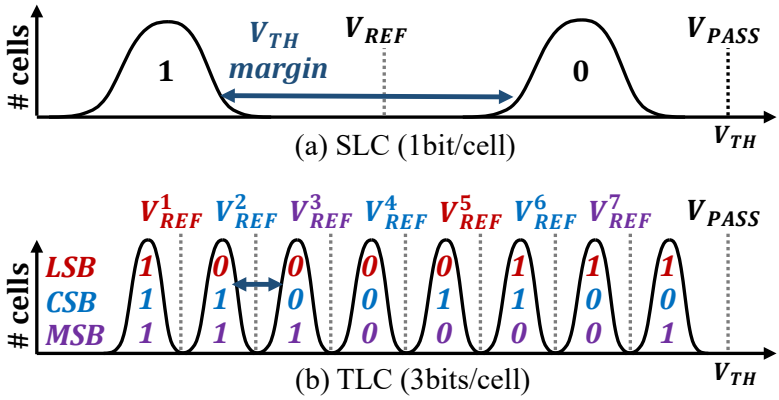


Figure 4: V_{TH} distributions of NAND flash memory.

(**0**). Discharging the WLs and BLs ensures that subsequent reads, writes, or erases are unaffected by the previously accessed block [15, 14, 16].

2.1.4 Multi-level Cell Technology

To increase the storage capacity, a flash cell can store multiple bits by adjusting its V_{TH} level more precisely, called *m-bit multi-level cell* technology. It stores m bits within a single flash cell by using 2^m distinct (V_{TH}) states (i.e., m is 2 and 3 for MLC and TLC flash memory, respectively). Figure 4 compares the V_{TH} distribution of a SLC with the V_{TH} distribution of a TLC that uses different 2^3 states to store 3 bits per cell, indicating each WL contains three pages called LSB, CSB, and MSB page.

While multi-level cell technology offers high storage capacity, it also brings performance degradation and reliability concerns. SLC NAND, for example, requires only a single sensing operation per read, enabling faster access times. In contrast, TLC NAND requires multiple sensing operations

to read a page, which increases read latency; for instance, reading a CSB page in TLC NAND demands three sensing operations (V^2_{REF}, V^4_{REF}, and V^6_{REF}). Furthermore, NAND flash is inherently prone to errors due to its imperfect physical characteristics. Over retention time after programming, cells gradually leak stored charge, causing shifts in the V_{TH}. These shifts can lead to misjudgment of the V_{TH} state during sensing, resulting in read errors. SLC NAND benefits from a large V_{TH} margin between states, contributing to its low error rate. In contrast, TLC NAND has a narrower V_{TH} margin, making it more susceptible to errors. Moreover, as cells undergo more program and erase (P/E) cycles, they gradually wear out, making charge leakage more likely and leading to increased errors. As a result, modern SSD controllers employ complex ECC engines to ensure data integrity before transferring the data to the host.

2.2 Read-Disturbance Problem

2.2.1 Read-Disturbance Mechanism

During a read operation, V_{PASS} is applied to non-target WLs to fully turn on the NAND string, suppressing unexpected bit errors in flash cells on the target WL. Although V_{PASS} is much lower than the program voltage, the flash cells on non-target WLs can be softly programmed by the electrical stress due to V_{PASS} (i.e., shift V_{TH} distribution to the right), as shown in Figure 5. Furthermore, when reads are performed repeatedly, the amount of stress accumulates (i.e., causing more V_{TH} shift), resulting in significant bit errors across all non-target WLs in a block, called read-disturbance er-

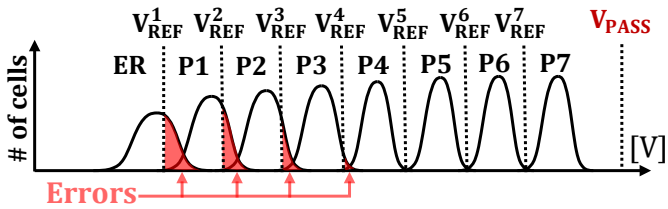


Figure 5: A V_{TH} distribution after read-disturbance.

ror [14].

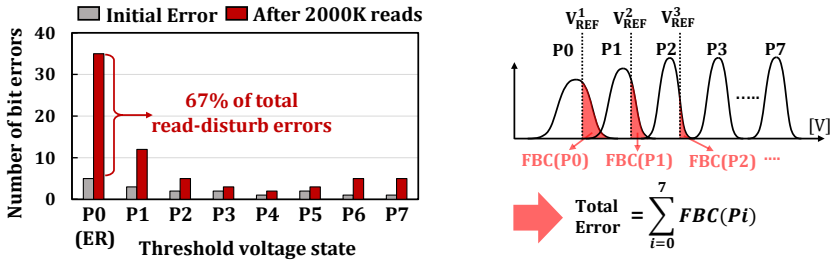
Based on the well-known Fowler–Nordheim (FN) tunneling equation [17], read-disturbance errors can be quantified as follows:

$$Error_{RD} \propto V_{pass}^2 \cdot exp(\frac{-1}{V_{pass}}) \cdot T_{read} \cdot N_{read} \tag{2.1}$$

where T_{read} and N_{read} are the read latency and the total number of read operations, respectively.

2.2.2 Read Disturbances in 3D Flash Memory

Since the read-disturbance error is proportional to three key factors, V_{PASS}, T_{read} and N_{read} (in Equation 2.1), it becomes more pronounced as the capacity of 3D flash memory increases. First, the more stacked WLs for high capacity, the higher the NAND string in Figure 1, resulting in higher resistance of a NAND string. Consequently, a higher V_{PASS} voltage is required to sufficiently turn on flash cells on the NAND string. Second, larger block size due to more stacked WLs in a block inevitably increases the total number of reads per block (i.e., N_{read}). Furthermore, higher m-bit multi-level cell flash memory requires much longer T_{read}. For example, SLC flash



(a) Per-state read-disturb error. (b) Per-state fail-bit count (FBC).

Figure 6: A breakdown of per-state errors on 3D TLC flash.

memory has a T_{read} of around 20 µs [18], while QLC flash memory requires 90 µs [19] for reading a page, significantly amplifying the read-disturb errors.

2.2.3 State-Dependent Read-Disturbance Errors

To better understand the read-disturb problem, we examine how read-disturb errors are distributed over different V_{TH} states. Figure 6(a) shows the breakdown of per-state bit errors after 2000K reads. Like 2D flash memory, read-disturb errors in 3D TLC flash memory occur in an *asymmetric fashion* [20, 21]. Repeated reads shift V_{TH} state to the right by soft-program effect, and such an effect makes some flash cells in the i'th V_{TH} state exceed its reference voltage V_{REF}^{i} (called *fail-bit*). The amount of read-disturb error can be calculated by counting the number of fail-bits, denoted as C_{FBC}, as shown in Figure 6(b). The soft-program effects are more pronounced in relatively lower V_{TH} states because the magnitude of V_{PASS} is not sufficient to shift the flash cell in higher V_{TH} states. Therefore, most read-disturb errors are highly localized in the lower V_{TH} states (e.g., ER and P1 states), while

other states show only a marginal increase in bit errors.

2.2.4 Reliability Management: Read Refresh

To protect data against read-disturbance errors, an SSD performs a read refresh (RR) procedure well before any uncorrectable errors occur; it reads all valid pages from the affected block, corrects the errors, and migrates (i.e., rewrites) the data to free blocks. A typical read-disturb management technique is based on a predefined constant C_{max} that is tailored to the number of reads that all blocks are guaranteed to tolerate. In other words, C_{max} is set to the minimum value over all C^B_{max} values where C^B_{max} is the maximum read count that the block B can tolerate. When the current number of reads to the block B reaches C_{max}, the RR procedure is invoked to avoid any potential data loss.

Although an RR procedure is essential to ensure data integrity against the read-disturb problem, unfortunately, it can cause significant performance and lifetime overheads in SSDs. The RR procedure requires extra internal data movements (i.e., reads and writes for data migration), which can conflict with host I/O requests. Furthermore, since data migration requires additional program and erase operations (i.e., P/E cycles), frequent invocations of an RR procedure inevitably lead to lowering the lifetime of SSDs.

2.3 Related Work

2.3.1 Read-Disturbance Management

Several prior system-level or device-level studies have focused on mitigating the lifetime and performance overheads incurred by RR. Prior works [22, 23, 24] have optimized data placement policies in an RR procedure to reduce the future RR invocations. Zhang et al. [22] proposed a strategy to disperse frequently-read pages, i.e., read hot data, widely across multiple free blocks. This approach optimizes the RR invocations by preventing the concentration of read hot data within only a few blocks. Liao et al. [23] proposed to distribute valid pages from an RR procedure using a data reallocation model that estimates the hotness of pages.

A few prior works [25, 21] have proposed to optimize read-disturb management by leveraging inter/intra-block variation. Zhao et al. [25] proposed to set different maximum tolerable read counts among blocks depending on their P/E cycles. Han et al. [21] proposed to classify pages within a block into several groups by page types (e.g., MSB/CSB/LSB in TLC flash memory) and migrate each based on different threshold values.

Prior works [24, 26] have proposed V_{PASS} scaling techniques for 2D flash memory to mitigate the read-disturb error. Ha et al. [24] proposed an integrated approach that programs a read-hot block with fine-grained control to form narrow V_{TH} distributions, and uses a lower V_{PASS} when reading a page in the read-hot block. Cai et al. [26] proposed to use a lower V_{PASS} when there is a sufficient ECC-capability margin in the target block. Since read disturb is strongly correlated with the level of pass-through voltage,

using low pass-through voltage can effectively suppress its negative impact.

2.3.2 In-Storage Processing

To address the limited external PCIe bandwidth of SSDs, numerous studies have proposed in-storage processing (ISP) solutions [7, 8, 9, 10, 11, 12, 27, 28, 29, 30, 31, 32, 33, 34] across diverse application domains such as pattern matching [35, 36], query processing [9, 10, 11], data analytics [12, 27], genome processing [7], graph processing [31, 32, 33], and DNN [28, 29, 30], which minimize data movement by processing data at the SSD controller and delivering only the results to the host.

A key advantage of ISP is its ability to leverage existing resources in the SSD controller, such as embedded processors and ECC engines, making it a cost-effective solution. However, existing ISP solutions cannot meet the high read bandwidth demands essential for emerging data-intensive applications such as LLM inference. In ISP solutions, data read from flash chips must be transferred to the SSD controller via flash channels, which use a multi-drop bus topology, as illustrated in Figure 1. This topology limits data transfers to one flash chip per channel at a time, creating a significant bottleneck at the flash channels. With SSDs typically offering up to 8 channels [37, 38, 39], each limited to 1.6–2.4 GB/s of bandwidth, ISP solutions can only achieve an aggregate bandwidth of 12.8–19.2 GB/s. In contrast, LLM inference often requires read bandwidths nearing 100 GB/s, rendering ISP solutions inadequate for such demanding requirements.

2.3.3 In-Flash Processing

To overcome the bottleneck caused by flash channels and fully utilize the internal bandwidth of SSDs, some studies have explored in-flash processing (IFP) solutions, such as graph neural network [1], vector search [2, 3], DNN [4, 5], and bitwise processing [6]. Modern SSDs are composed of multiple flash chips (e.g., 16 chips in a 1-TB SSD), whose aggregate read bandwidth, referred to as internal bandwidth, far surpasses the constrained bandwidth of flash channels. By enabling computations to occur directly within the flash chips through dedicated in-flash accelerators, SSDs can effectively leverage their high internal bandwidth to support data-intensive applications. For example, single-batch LLM inference primarily involves GEMV, which inherently reduces data size by transforming input matrices (e.g., model parameters) into smaller output vectors (i.e., token embedding vectors). This high data reduction characteristic makes LLM inference particularly well-suited for IFP. For instance, a large matrix (e.g., 175-MiB) stored across flash chips can be processed using its high internal bandwidth, with only the significantly smaller output vector (e.g., 16-KiB) transferred over the limited flash channel bandwidth.

While prior works have successfully demonstrated the potential of IFP solutions, they have largely focused on integrating IFP into applications without adequately addressing practical challenges, which will be discussed in Chapter 3. The IFP-specialized techniques proposed in this work for improving bandwidth and reliability can complement existing approaches, offering an orthogonal enhancement to previous work.

Chapter 3

Key Challenges in In-Flash Processing

In-flash processing (IFP) is a highly promising paradigm for accelerating data-intensive applications; however, its practical adoption requires overcoming three fundamental challenges, which are deeply rooted in the capacity-oriented evolution of modern NAND flash memory. Over the past decades, NAND flash memory has aggressively increased memory density through technologies such as multi-level cell and 3D stacking in order to meet the paramount requirement of storage systems: capacity. While these advances have been highly effective in scaling capacity, they have come at a significant cost to flash reliability and performance. This degradation in reliability and performance of modern NAND flash memory poses particularly critical challenges for IFP because, unlike conventional SSDs, IFP executes computation directly inside flash chips. In this chapter, we introduce each of the three key challenges and systematically analyze their impact on the efficiency of IFP.

3.1 Diminishing Read Bandwidth

3.1.1 Capacity Scaling of NAND Flash Memory

Figure 7(a) illustrates the cell-array organization of earlier-generation NAND flash memory. A cell array typically consists of several tens of word-

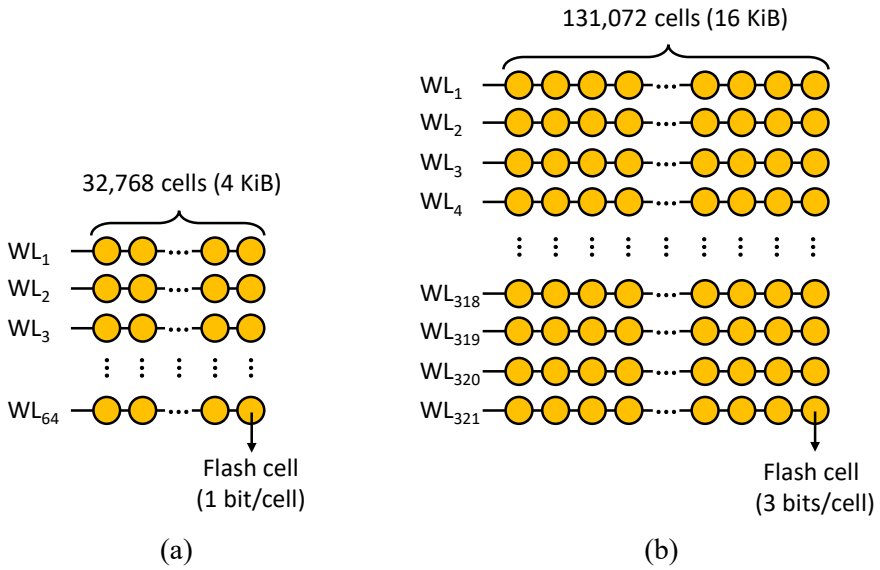
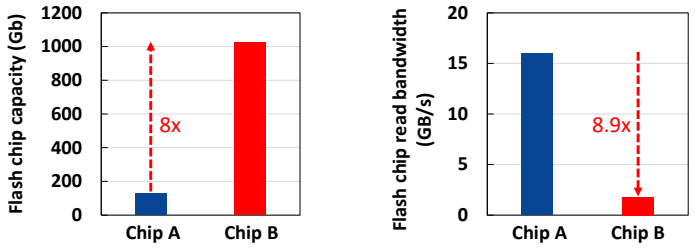


Figure 7: Comparison of the cell-array organization in (a) earlier-generation and (b) modern high-density NAND flash memory.

lines, where each wordline connects to approximately 32,768 cells (corresponding to a 4 KiB page), and each flash cell stores a single bit of information. In contrast, Figure 7(b) shows the cell-array organization of recent NAND flash memory. With the transition to 3D stacking, the number of wordlines has increased to several hundreds. Also, to reduce the area overhead of peripheral circuits such as row decoders, the number of cells connected to each wordline has increased by approximately 4x. Moreover, multi-level cell technologies have been adopted, enabling each cell to store three or even four bits of data. While these architectural changes have been highly effective in improving storage density, they have substantially increased flash read latency. This increase is primarily due to (1) the significantly larger number of flash cells that must be driven and sensed during

each read operation, often more than 20x compared to earlier generations, and (2) the increased complexity of read procedures required for multi-bit cells, which rely on multiple sensing steps.

Figure 8 compares the capacity and read bandwidth of two flash chips: Chip A [40], which follows the organization of earlier-generation NAND flash memory, and Chip B [41], which represents modern high-density NAND flash memory. Driven by capacity-oriented design choices, Chip B provides approximately 8x higher capacity than Chip A within the similar chip area. However, for the reasons discussed above, Chip B achieves 8.9x lower read bandwidth than Chip A. This bandwidth degradation may not be a critical issue for conventional SSDs, since their performance is often bounded by the external PCIe interface and can therefore partially mask the reduction in internal flash bandwidth. In contrast, IFP executes computation directly inside flash chips and thus relies heavily on internal flash read bandwidth, making its performance directly constrained by such bandwidth degradation. As capacity-oriented flash scaling is expected to continue, flash read bandwidth is likely to decline further.



(a) Capacity (b) Read bandwidth

Figure 8: Comparison of (a) flash-chip capacity and (b) read bandwidth between Chip A and Chip B.

3.1.2 Limitations of IFP for Read-Intensive Workloads

The diminishing read bandwidth of modern NAND flash memory fundamentally limits the ability of IFP to support diverse workloads, particularly those with stringent read-bandwidth demands such as on-device LLM inference. For example, on-device LLM inference exhibits extremely low arithmetic intensity (only 1–2 operations per byte) [42, 43], primarily due to their single-batch processing characteristic, making inference speed highly dependent on the read bandwidth of model parameters. Additionally, the autoregressive nature of LLM inference necessitates reading the entire model parameters for each token generation step [42, 43, 44]. This dependency places an upper bound on the token generation rate, expressed in tokens per second (tokens/s), as follows:

$$\text{max. tokens/s} \leq \frac{\text{read bandwidth } (\text{GB/s})}{\text{model size } (\text{GB})}. \tag{3.1}$$

Although IFP enables SSDs to fully utilize their internal bandwidth, it still falls significantly short of the bandwidth required to meet the commonly desired throughput for LLM inference. Figure 9 shows the internal bandwidth of TLC SSDs. A 1-TB TLC SSD with 16 flash chips [37, 39], a popular configuration in personal devices like laptops and smartphones, provides an internal bandwidth of only 25.6 GB/s, primarily due to the diminishing read bandwidth of modern NAND flash memory discussed above. From equation 3.1, this limitation means that when running inference on a 30B model with 30 GB of parameters, a 1-TB SSD only achieves less than 0.85 tokens/s even with IFP. One way to increase internal bandwidth is to

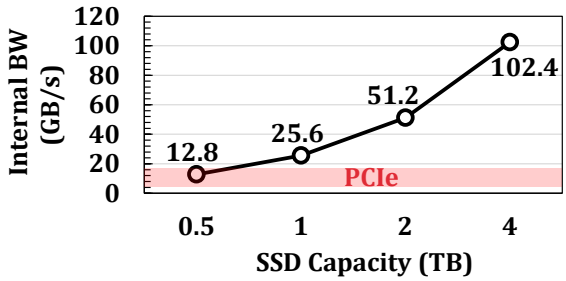


Figure 9: Internal bandwidth of SSDs.

scale up SSD capacity by adding more flash chips (e.g., 4-TB capacity). However, this significantly raises the cost of edge devices, undermining the benefit of IFP.

3.2 High Read Disturbance

3.2.1 Read Disturbance in Modern SSDs

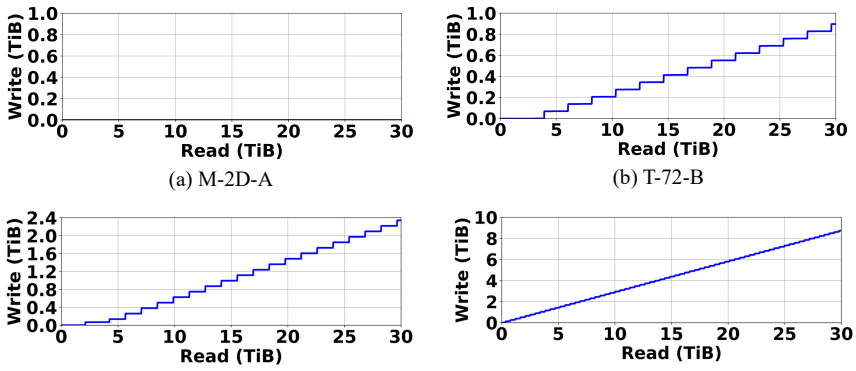
As discussed in Section 2.2, NAND flash memory suffers from read disturbance, where each read operation gradually introduces bit errors into the stored data. To recover from these disturbance-induced errors and maintain data integrity, SSDs periodically perform an internal maintenance task called read refresh (RR), which migrates and rewrites highly disturbed data to restore reliability. While RR is essential for ensuring correct operation, it inevitably incurs additional internal writes due to data migration, which degrades both SSD lifetime and performance.

To quantify the impact of read disturbance, we evaluated how severely SSD lifetime and performance are degraded by RR (i.e., RR overhead) using commercial SSDs.

Table 1: Detailed specifications of 11 SSDs for evaluation.

Name	Flash type	No. of stacked wordlines per block	Block size (MiB)	Capacity	Interface	Vendor
M-2D-A	2D MLC	N/A	8	250 GB	SATA 6Gb/s	A
T-176-A	3D TLC	176	33	500 TB	SATA 6Gb/s	A
Q-176-A	3D QLC	176	44	500 GB	SATA 6Gb/s	A
T-72-B	3D TLC	72	14	500 GB	SATA 6Gb/s	B
T-64-A	3D TLC	64	12	1 TB	NVMe 4GB/s	A
T-176-A	3D TLC	176	33	1 TB	NVMe 8GB/s	A
Q-176-A	3D QLC	176	44	1 TB	NVMe 8GB/s	A
T-128-C	3D TLC	128	24	1 TB	NVMe 8GB/s	C
Q-232-C	3D QLC	232	58	1 TB	NVMe 8GB/s	C
T-64-D	3D TLC	64	12	500 GB	NVMe 4GB/s	D
T-162-D	3D TLC	162	57	1 TB	NVMe 8GB/s	D

Evaluation Setup. Table 1 lists the detailed specifications of eleven SSDs used in our evaluations. We selected 11 SSDs with different multi-level cell technologies and block sizes from three different SSD vendors. The name X-Y-Z SSD means that it employs (X)LC flash memory with the number of **(Y)** stacked WLs manufactured from **(Z)** vendor. For example, T-176-A SSD and Q-176-A SSD are made by 3D flash memories from vendor **A** with the same number of WL stacks but different multi-level cell technologies (i.e., TLC and QLC flash memory, respectively). For each SSD under test, we used a single-threaded FIO [45] configured to perform 4-KiB random reads with a queue depth of 32. To assess the impact on SSD lifetime, we tracked internal write activity during read operations (i.e., RR-induced writes) by monitoring S.M.A.R.T. logs [46]. To evaluate the read latency impact, we collected the read latency for every 4 ms. To examine the effect of working set size, evaluations were conducted over two LBA ranges: 250 GB and 64 GB. Prior to each test, a full-drive write pass was performed across the entire LBA range to eliminate any residual effects from previous tests.



(a) M-2D-A (b) T-72-B

(c) T-176-A (d) Q-176-A

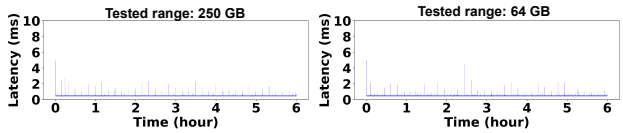
Figure 10: A comparison of RR-induced writes in four SSDs.

Impact on Lifetime. Figure 10 shows the cumulative internal writes triggered by the RR procedure as the total amount of reads increases for each SSD. The evaluation results show that RR-induced writes exponentially amplify in high-capacity SSDs. For instance, the 2D MLC SSD (M-2D-A) exhibits no RR-induced writes, as shown in Figure 10(a). In contrast, the 3D TLC SSDs, shown in Figures 10(b) and (c), suffer from a large number of RR procedures as reads accumulate, leading to nearly 0.9-TiB and 2.4-TiB of additional RR-induced writes after 30-TiB of reads. Such negative impact is even more severe in the Q-176-A SSD, which incurs approximately 8.2-TiB of RR-induced writes, as shown in Figure 10(d). Assuming a 200 TBW SSD endurance [47, 48] and a sustained read throughput of 100 MB/s, the T-176-A and Q-176-A are expected to last less than 263 days (-85%) and 77 days (-95%), respectively, under read-intensive workloads compared to the 5-year warranty. These results highlight the negative impact of the RR procedure on SSD lifetime, especially for modern high-capacity flash tech-

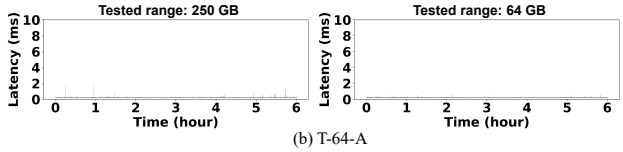
nologies.

Impact on Read Latency. Figure 11 shows the read latency profiles of each SSD in read-intensive environments. While reads are repeated, the read latency of most tested SSDs significantly fluctuates over time due to the time-consuming RR procedure. From the evaluation results, we quantitatively analyze the impact of capacity-centric flash techniques on RR overhead and reveal why optimizing the RR procedure is important in modern high-capacity SSDs.

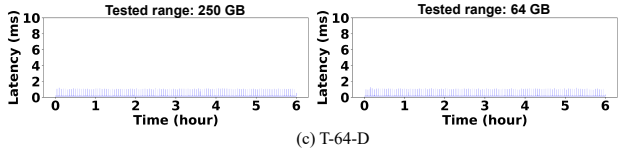
To examine the impact of the block size on RR invocations, we compare the read latency profiles of two 1-TB SSDs, T-64-A and T-176-A. Both SSDs employ high-capacity 3D TLC flash memory, but due to the different stacked WLs, the block size of T-176-A is 2.75 times larger than that of T-64-A. As depicted in Figures 11(b) and 11(f), SSDs with higher WL stacks (i.e., larger block size) exhibit more latency degradation compared to SSDs with fewer WL stacks (i.e., smaller block size). For SSDs with large block sizes, it is likely that multiple read requests can be served from one single large block (i.e., increase N_{read} per block). Consequently, even at low read counts, read-disturb errors sharply increase, which, in turn, leads to more frequent RR invocations. Moreover, a larger block size requires more data migration (i.e., data copy for more WLs) during each RR procedure, further amplifying the performance overhead. Similarly, the evaluation results for T-64-D and T-162-D, as shown in Figures 11(c) and 11(e), also demonstrate the risk of large block size. In addition, more severe fluctuations in read latency are observed in a smaller working size. It indicates that in a smaller working size, read requests are concentrated into fewer blocks, accelerating



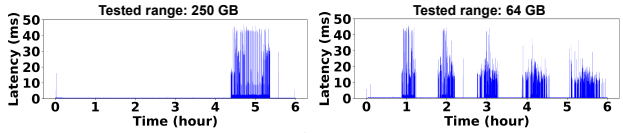
(a) M-2D-A



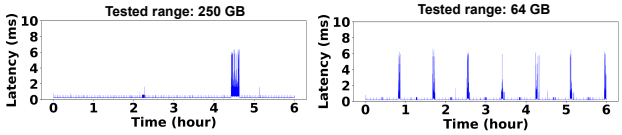
(b) T-64-A



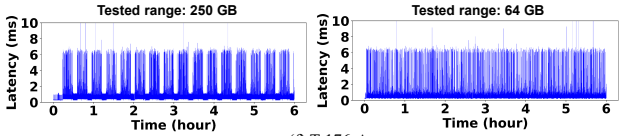
(c) T-64-D



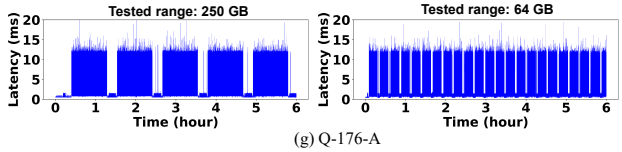
(d) T-128-C



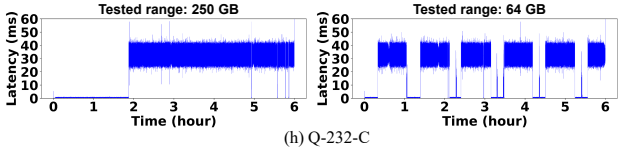
(e) T-162-D



(f) T-176-A



(g) Q-176-A



(h) Q-232-C

Figure 11: A comparison of read latency in eight SSDs.

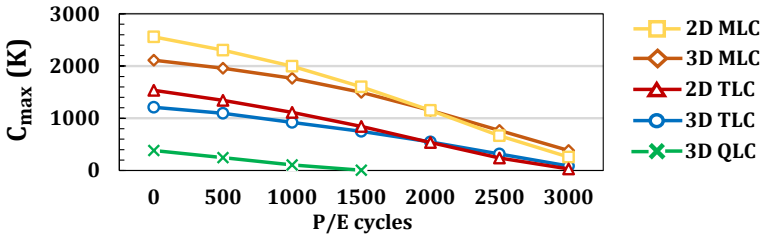


Figure 12: Changes in C_{max} values in different flash memories.

read-disturb errors of the target blocks.

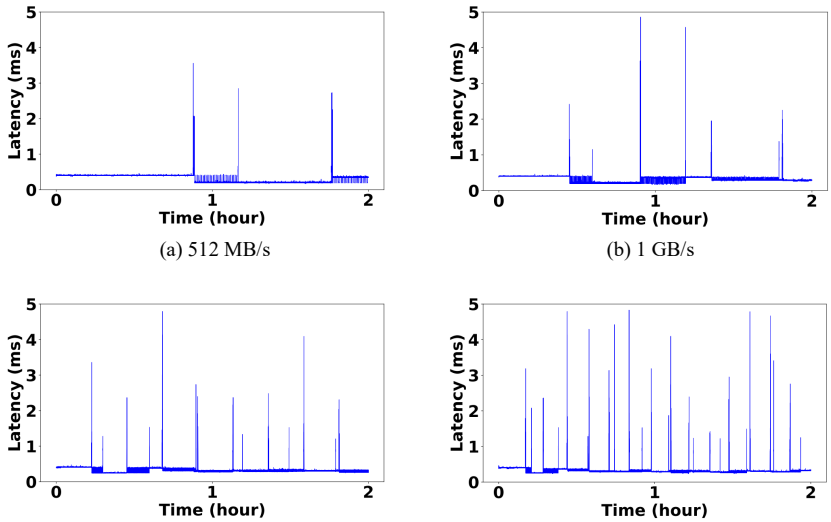
Figures 11(f) and 11(g) show the impact of m-bit multi-level cell technologies on RR invocations. A read latency comparison for T-176-A and Q-176-A SSDs indicates that SSDs with higher m-bit multi-level cell flash memory (e.g., QLC flash memory) suffer from more RR invocations than SSDs with lower m-bit multi-level cell flash memory (e.g., TLC flash memory). As evaluations using real flash chips in Figure 12, higher m-bit multi-level cell flash memory should set C_{max} sufficiently low because of its reduced error margin, resulting in more frequent RR invocation. For example, the C_{max} of QLC flash chip is 3.15 times and 5.5 times lower than that of TLC and MLC flash chip, respectively. Furthermore, since higher m-bit multi-level cell flash memory has poorer read and write latency, it requires a longer time to complete the RR procedure, leading to more significant fluctuation in read latency.

3.2.2 Read Disturbance in IFP-enabled SSDs

The lifetime and performance degradation caused by RR is particularly problematic for IFP-enabled SSDs. First, IFP is primarily designed to

accelerate read-intensive workloads such as LLM inference, vector similarity search, and graph neural networks, which issue significantly more reads than general workloads. This high read volume amplifies read-disturbance stress and consequently increases the frequency of RR invocations. Second, unlike traditional DRAM+SSD architectures that rely heavily on DRAM caching to reduce repeated flash reads, IFP leverages high internal bandwidth to execute computation inside flash chips and transfers only the final results to the host. In other words, IFP often processes datasets directly from flash without caching them in DRAM, which increases the number of flash reads required for the same workload. Third, whereas conventional SSDs are limited by the external PCIe bandwidth (typically 4–15 GB/s), IFP activates all flash chips concurrently to exploit the SSD's internal bandwidth. Consequently, IFP exposes flash memory to substantially higher read-disturbance stress within a shorter time window than traditional SSD operation.

To quantitatively estimate how frequently RR may be triggered under IFP, we evaluate the impact of RR on SSD behavior under different levels of read intensity. Figure 13 shows the read-latency fluctuations induced by RR when an SSD repeatedly performs sustained sequential reads at 512 MB/s, 1 GB/s, 2 GB/s, and 4 GB/s, respectively. We use repetitive sequential reads to emulate the access pattern of LLM inference, which repeatedly reads model weights for each token-generation step. As expected, RR is triggered more frequently as the read intensity increases. At a sustained read bandwidth of 512 MB/s (Figure 13(a)), only two noticeable latency spikes are observed, indicating two RR invocations. However, as the sustained read bandwidth doubles, the RR frequency increases almost linearly: at 1 GB/s, 2



(a) 512 MB/s (b) 1 GB/s

(c) 2 GB/s (d) 4 GB/s

Figure 13: A comparison of read latency under different read intensities.

GB/s, and 4 GB/s, we observe four, eight, and fourteen RR-induced latency spikes, respectively (Figures 13(b)–(d)). These results strongly suggest that RR will become significantly more frequent under IFP. As discussed earlier, IFP typically processes datasets directly from flash without relying on DRAM caching and activates all flash chips concurrently to maximize internal bandwidth, resulting in extremely high effective read bandwidth (e.g., tens to hundreds of GB/s). Consequently, IFP-enabled SSDs are expected to experience substantially higher read-disturbance stress and incur much heavier RR overhead than conventional SSD operation. Overall, this analysis demonstrates that mitigating read disturbance, and reducing the resulting RR overhead, is a first-order challenge for the practical deployment of IFP.

3.3 High Implementation Overheads

3.3.1 Root Causes of High Implementation Overheads: PPA Inefficiency and ECC Complexity

To enable data processing inside flash chips, IFP typically requires integrating two major logic components into the flash chip peripheral circuitry: (1) computation logic for in-flash processing and (2) an ECC decoder to recover data corrupted by flash errors. These additional components can substantially increase both chip area and power consumption, making implementation overhead a critical barrier to practical IFP deployment.

First, the peripheral circuitry of NAND flash chips is manufactured under severe process constraints that lead to poor power-performance-area (PPA) efficiency compared to conventional CMOS logic. Similar to DRAM, flash peripheral circuits are typically fabricated using relatively large process nodes and a limited number of metal routing layers. This is largely due to the unique fabrication flow of 3D NAND, where peripheral circuits are manufactured first and the flash memory array is subsequently built on top. To protect peripheral circuits from the high thermal budget required for fabricating the flash array, manufacturers often employ conservative process technologies. As a result, implementing the same logic in flash peripheral circuits can incur significantly higher area and power overhead than implementing it in modern ASICs built with advanced CMOS processes.

Second, NAND flash memory inherently requires powerful and complex error-correction mechanisms. Unlike DRAM, flash memory exhibits high raw bit error rates, often resulting in tens of bit errors per kilobyte.

To ensure correctness, modern SSD controllers therefore employ sophisticated ECC engines such as low-density parity-check (LDPC) decoder. However, because IFP performs computation inside flash chips, it cannot rely on the controller-side ECC engine. Instead, it must incorporate on-chip ECC decoder to ensure reliability. Unfortunately, the high complexity of ECC decoder and the limited PPA efficiency of flash peripheral circuits make on-chip ECC decoders extremely costly in both area and power. In particular, LDPC decoders require extensive wiring and high routing complexity, which may be infeasible to implement given the limited metal layers available in flash chip peripheral circuitry.

3.3.2 Quantitative Evaluation of On-Chip ECC Overheads

One approach to mitigating the implementation overhead of ECC decoders is to target error-resilient applications, as explored in prior IFP studies [3, 5, 4]. Unfortunately, many emerging data-intensive applications, such as LLM inference and graph neural networks, are highly vulnerable to even small bit errors. To examine the impact of raw bit error rate (RBER) on LLM inference, we evaluated how the model accuracy changes as the number of error bits in model parameters increases by injecting bitflip errors into model matrices.[1] Figure 14 shows the average model accuracy as RBER increases, revealing notable sensitivity to even extremely low RBER ranges. For example, the LLaMA-3 8B model (INT8-quantized) experiences an accuracy

[1]For accuracy evaluation on HellaSwag [49], all models were quantized to INT8. We then injected RBER-based bit-flips probabilistically, but only in the QKV projection and FFN weight matrices (see Figure 41).

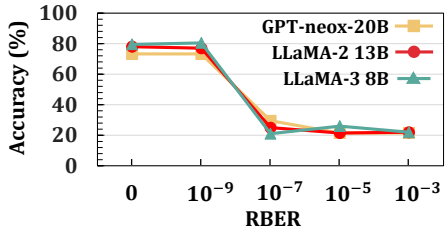
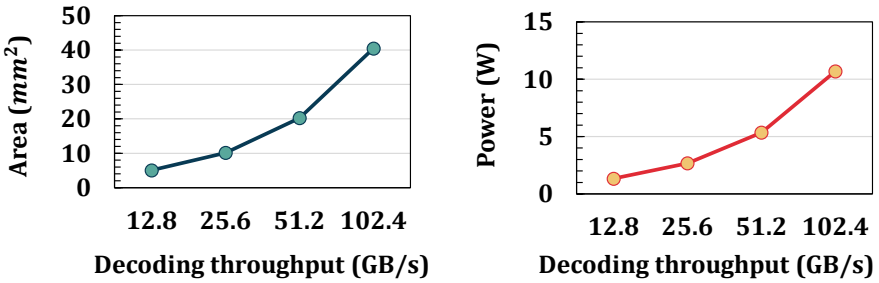


Figure 14: Impact of RBER on inference accuracy.

drop of over 60% at an RBER of 10^{-7}, which is far below the typical RBER ranges of NAND flash memory ($> 10^{-3}$). This highlights that LLM inference is not inherently error-tolerant, making ECC decoder within the flash chip (i.e., on-chip ECC) indispensable to ensure the inference accuracy.

While on-chip ECC is essential for ensuring correct operation, it introduces significant implementation overhead. This overhead arises because, as discussed earlier, ECC decoders must be implemented within the flash-chip peripheral circuitry, which suffers from poor PPA efficiency, and must simultaneously deliver high decoding throughput while providing strong error-correction capability to tolerate the high raw bit error rate of NAND flash memory. For conventional off-chip ECC decoders implemented at the SSD controller, error correction is primarily performed on data delivered to the host via the PCIe interface, resulting in the decoding throughput requirement is dependent on the external bandwidth. In contrast, on-chip ECC decoders perform error correction directly within each flash chip, requiring the total correction throughput across all flash chips to match the SSD's internal bandwidth. Since the overhead of ECC scales linearly with throughput, this imposes a substantial burden on SSDs.



(a) Area overhead (b) Power overhead

Figure 15: PPA impact of ECC decoder.

Figure 15 illustrates the estimated area and power consumption of a BCH-based ECC decoder at increasing decoding throughputs. The ECC decoder is configured to correct 50-bit errors per 1-KiB. To achieve 102.4 GB/s throughput, the cumulative on-chip ECC decoders demand 40.12 mm^2 of silicon area and consume 10.694 W, which far exceeds the power budget of consumer-grade SSDs (6-8 W [37, 39]). These results emphasize the substantial implementation overhead of high-throughput yet complex on-chip ECC, presenting significant challenges of IFP.

Chapter 4

Mitigating Read Disturbance via Dynamic Error-Aware Refresh

4.1 Limitations of the Existing Solutions

Since RR is an essential procedure to recover data from read-disturb errors, prior works [22, 23, 25, 21, 26, 50] have focused on alleviating its negative impact using techniques such as data reallocation [22, 23, 25, 21], voltage scaling [26], and buffer management [50]. However, no existing study has explored how to invoke RR in a *timely manner* to minimize the RR invocations. Similar to other memory technologies (e.g., DRAM), modern 3D flash memory exhibits significant process variation, resulting in widely differing physical characteristics across WLs [51, 52]. Unfortunately, previous studies rely on a predetermined fixed C_{max}, defined as the C_{max}^{WL} of the worst WL in the worst block, to trigger RR. By ignoring the high process variability of 3D flash memory, the conservative C_{max} setting in the existing approach leads to premature and unnecessary RR invocations.

Analysis of C_{max}^{WL} Variations. To quantitatively understand the process variability in read-disturb errors, we examine the C_{max}^{WL} variation between WLs using 160 3D TLC flash chips (see Section 4.3 for details on our characterization methodology). For obtaining statistically significant results, we

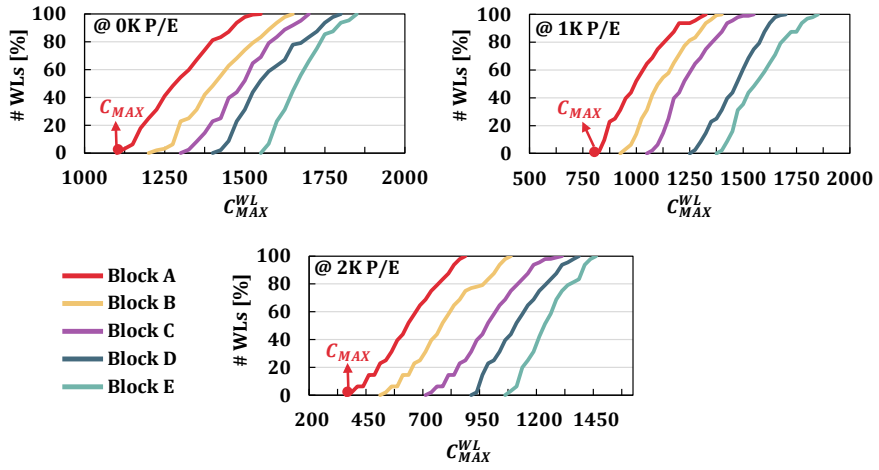


Figure 16: Per-WL C_{max}^{WL} distributions over different P/E cycles.

measure C_{max}^{WL} for 3,686,400 WLs randomly selected from 160 flash chips (19,200 blocks in total). Figure 16 shows the cumulative distribution function (CDF) of C_{max}^{WL} across 5 selected blocks (blocks A $\sim$ E) at different P/E cycles. Our results reveal substantial C_{max}^{WL} variation across WLs, even at the same P/E cycle. For example, at 0K P/E cycle, the average C_{max}^{WL} gap between the best block and the worst block is as much as 250K reads. Furthermore, even in the same block, the C_{max}^{WL} value of the best WL is about 1.42x higher than the worst WL. The C_{max}^{WL} variation becomes more severe as P/E cycles increase (i.e., flash memory gets aged). At the 2K P/E cycle, the gap between the best and worst C_{max}^{WL} widens up to 4.14x.

From our characterization results, we derive two key conclusions. First, due to the significant variation in C_{max}^{WL} between WLs, the existing block-level RR management scheme based on a single C_{max} value is no longer practical for managing the read-disturb problem. While previous techniques

may be reasonable for 2D or earlier 3D NAND flash memory, where blocks are smaller (e.g., 4-MiB [53]) and process variation is less significant [51, 52], they become highly inefficient for modern 3D NAND flash memory with ever-increasing capacity. Second, if it is possible to accurately estimate the actual C_{max}^{WL} of each WL and only highly disturbed WLs are selectively refreshed, unnecessary RR invocations can be minimized, providing an effective solution to mitigate RR-induced overheads.

4.2 Overview of DEAR

We propose DEAR *(Dynamic Error-Aware Refresh)*, a novel read-disturb management scheme designed to improve SSD lifetime and performance by effectively reducing RR invocations. Figure 17 shows how the RR procedure can be optimized under the DEAR.

The key idea of DEAR is simple. Unlike existing solutions that refresh all blocks when their read count (RC) reaches a fixed C_{max} (all WLs are refreshed at 100 reads, ❶ in Figure 17), DEAR dynamically allocates near-optimal C_{max}^{WL} values to each WL. Specifically, DEAR determines the maximum number of reads each WL can tolerate before its errors exceed the ECC correction capability (ECC_{CAP}). Using per-WL C_{max}^{WL} values, DEAR performs RR at the WL level, selectively refreshing only the WLs that are actually highly disturbed, rather than refreshing entire WLs in a block (only four WLs are refreshed at 300 reads, ❷). By doing so, DEAR fully leverages the reliability potential of each WL, significantly minimizing RR invocations in modern SSDs.

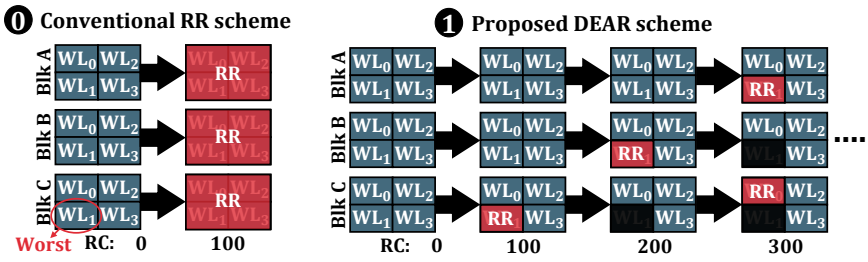


Figure 17: An illustrative example of the DEAR scheme.

To precisely determine C_{max}^{WL} values, DEAR *monitors* the dynamic error status of the WLs at runtime. C_{max}^{WL} is defined as the maximum number of reads a WL can tolerate before its error rate surpasses the ECC_{CAP}. DEAR ensures timely RR management by continuously tracking the actual error status of individual WLs and invoking the RR procedure only when a WL's error rate approaches the ECC_{CAP}.

Key Challenge. While the key idea of DEAR is straightforward, tracking the error status of each WL at runtime can introduce significant performance overhead in SSDs. In conventional SSDs, errors can be detected by reading data and applying error correction through the ECC decoder [54]. Therefore, a straightforward implementation of DEAR might involve reading and correcting all valid WLs in a block to identify those approaching the ECC_{CAP}. However, the large size of 3D flash block makes this approach impractical. In modern 3D NAND flash memory, a single block can exceed 120 MiB [41], which means that one round of error monitoring for all WLs in a block may take up to 337 ms—an unacceptably high latency. To address the challenge, DEAR employs the SEE (Swift Error Estimator), a mechanism specifically designed for accurate and efficient per-WL error es-

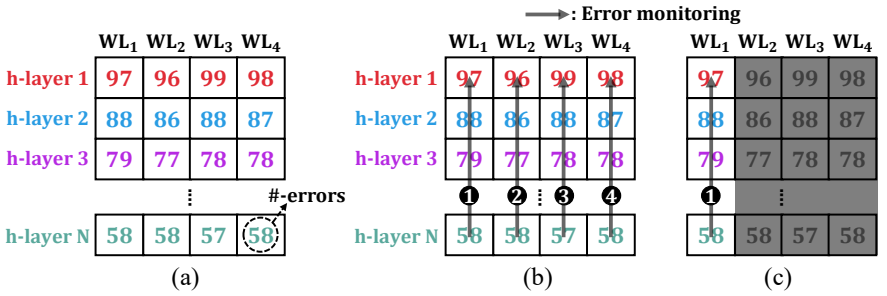


Figure 18: (a) Error similarity within a 3D block. Comparison of (b) straightforward error monitoring and (c) SAM.

timation. SEE incorporates three optimizations: (1) Similarity-Aware Monitoring, (2) Fail-bit-assisted Error Monitoring, and (3) Block-wise Periodic Monitoring, each of which is extensively validated through our real-device characterization study (Section 4.3).

Optimization 1: Similarity-Aware Monitoring (SAM). SAM is a simple yet highly effective technique that leverages the inherent error similarity in 3D flash memory to minimize the number of WLs to be monitored. As illustrated in Figure 1, a 3D flash block consists of vertically stacked h-layers, each containing four WLs (WL^1, WL^2, WL^3, and WL^4), separated by SSL transistors. Based on extensive device characterization, we observe that WLs within the same h-layer exhibit nearly identical error behavior, as illustrated in Figure 18(a). Figures 18(b) and 18(c) show how DEAR exploits this observation: by monitoring only the leading WLs (i.e., WL^1) of each h-layer, DEAR can reliably infer the error status of the remaining WLs in the same h-layer. Hence, SAM can reduce the number of WLs for error-checking by 4x, significantly lowering overhead while maintaining the accuracy of per-WL error estimation.

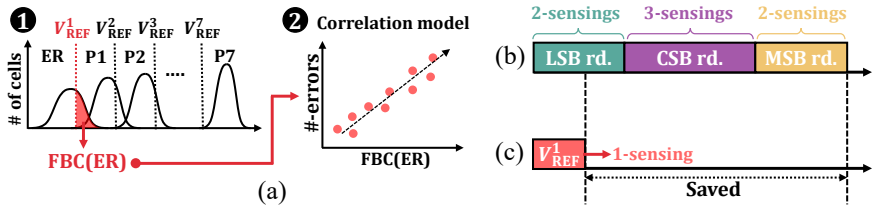


Figure 19: (a) Correlation between C_{FBC} and errors. Comparison of (b) ECC-based error monitoring and (c) FEM.

Optimization 2: Fail-bit-assisted Error Monitoring (FEM). To optimize DEAR in a latency-efficient way, we design FEM by exploiting asymmetric per-state characteristics in read-disturb error shown in Figure 6(a). Figure 19(a) illustrates the overall workflow of FEM. As demonstrated in Section 2.2.2, the root cause of read-disturb errors is an unexpected V_{TH} shift in the specific states, which is directly captured by the fail-bit count, C_{FBC}. C_{FBC} can be a reliable proxy for read-disturb errors because a higher C_{FBC} indicates a more severe V_{TH} shift, which means a higher number of read-disturb errors. To check the error in a WL, FEM measures only the C_{FBC} in the ER state (i.e., the C_{FBC} in the leftmost erased V_{TH} state in Figure 6), $C_{FBC}(\mathrm{ER})$, which is the most sensitive to read-disturb error, without requiring full page reads. Using an offline-built correlation model between $C_{FBC}(\mathrm{ER})$ and actual flash error, FEM accurately estimates the error status of the target WL.

Figures 19(b) and 19(c) highlight the performance benefits of FEM. Conventional error monitoring relies on the ECC decoder, which requires reading the entire WL—involving three page reads in TLC NAND, as illustrated in Figure 19(b). For example, reading an LSB page requires two

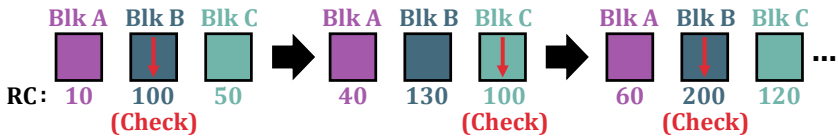


Figure 20: An illustrative example of BPM when the monitoring interval C_{INT} is set to 100 reads.

sensings (V^1_{REF} and V^5_{REF} in Figure 5), whereas reading a CSB page requires three sensings (V^2_{REF}, V^4_{REF}, and V^6_{REF}). In contrast, FEM requires only a single sensing operation and no ECC decoding, reducing the monitoring latency by nearly 7x, as shown in Figure 19(c). The detailed methodology for measuring $C_{FBC}(ER)$ with V^1_{REF} sensing is described in Section 4.4.

Optimization 3: Block-wise Periodic Monitoring (BPM). To further reduce the monitoring overhead, we introduce a technique, BPM, that minimizes the frequency of error monitoring. Ideally, the error monitoring task should be performed after every read, as the read-disturb effect accumulates whenever the stored data is accessed. However, this approach is infeasible due to the excessive monitoring overhead. Instead, DEAR performs error monitoring at regular intervals on a per block basis to control the frequency of monitoring. Figure 20 illustrates an example of how BPM operates. BPM enables DEAR to periodically perform error monitoring only for specific points where a block read count reaches a predefined number of reads, denoted by C_{INT}. In the example, C_{INT} is set to 100 reads, so the monitoring task is triggered only when the block's read count reaches a multiple of 100. This approach is reasonable from the fact that the read-disturb error gradually increases with read counts, not suddenly rising. By selecting an optimized value for C_{INT}, DEAR effectively reduces monitoring frequency

and amortizes the performance impact over time.

Although BPM can reduce monitoring frequency, it must also address potential reliability risks. Since BPM triggers error monitoring tasks only at every C_{INT} reads, DEAR should ensure that the accumulated read-disturb errors between the monitoring points do not cause data loss. In other words, even if a WL is under the ECC_{CAP} at the current monitoring point, its errors may exceed ECC_{CAP} by the next monitoring point. To ensure safe operation, DEAR proactively lowers the RR threshold (i.e., error value to invoke RR procedure) from ECC_{CAP} to a slightly more conservative value, E_{TH}, which accounts for the worst-case error accumulation during the monitoring intervals. Taking advantage of the safety margin ($ECC_{CAP} - E_{TH}$), DEAR guarantees that each WL remains within ECC_{CAP} until the next monitoring point, ensuring data integrity, although with a slight reduction in RR efficiency.

4.3 Design of Swift Error Estimator

In this section, we conduct an extensive real-device characterization study to validate our key optimization techniques in Section 4.2.

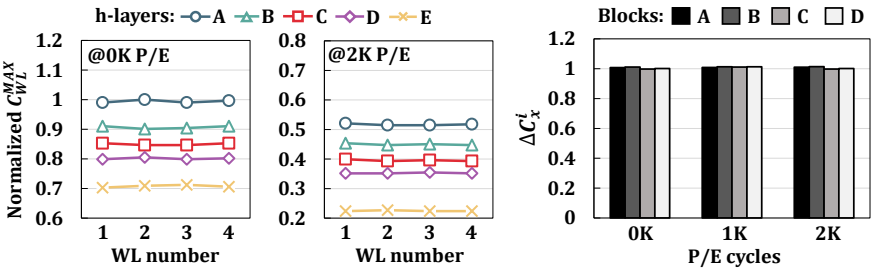
Characterization Methodology. We characterize 160 real 3D TLC flash chips using an FPGA-based testing platform with a custom flash controller and a temperature controller. The flash controller can perform not only basic NAND commands (i.e., for read, program, and erase operations) but also low-level test commands such as `GET/SET FEATURE` commands [55]. To minimize the potential distortions in our results, we evenly select 120

blocks from each chip at different physical locations and test all WLs (a total of 3,686,400 WLs) in each selected block. We set the ECC_{CAP} to 60 bits per 1-KiB [56]. To avoid pattern dependency all WLs are programmed with pseudo-randomized data [57]. Each test scenario follows the JEDEC standard [58] common in the memory industry.

4.3.1 Intra-Layer Similarity for SAM

To validate the feasibility of SAM, we analyze the C_{max}^{WL} values of the WLs within the same h-layer in a 3D NAND block. To quantitatively examine the degree of intra-layer similarity, we define a metric, ΔC_x^i, under a given (i, x). ΔC_x^i is defined as a ratio between the maximum C_{max}^{WL} and the minimum C_{max}^{WL} of i-th h-layer under x P/E cycles. As ΔC_x^i gets closer to 1, we can conclude that there exists stronger inter-layer similarity. Figure 21(a) shows the C_{max}^{WL} values of five representative h-layers selected from the tested h-layers. The measured C_{max}^{WL} is normalized over the h-layer that has the best C_{max}^{WL}. Unlike the pronounced inter-layer error variation observed along the vertical direction in the 3D NAND block, WLs in the same h-layer exhibit nearly identical C_{max}^{WL} values. Figure 21(b) shows that the intra-layer similarity is consistently maintained over different blocks as well under varying P/E cycles. In all our tested 921,600 h-layers, virtually all the ΔC_x^i values are close to 1 (e.g., ΔC_x^i among four WLs on each h-layer is less than 0.85% across all tested layers), regardless of the flash aging conditions.[1]

[1]This consistency can be attributed to two key factors. First, WLs within the same h-layer have similar physical characteristics (e.g., identical channel hole diameters) [52], as shown in Figure 2(b). Second, WLs on the same h-layer are physically connected through a common WL gate, indicating that all flash cells in WLs on the same h-layer lie in the same voltage conditions during read operation (Figure 1(a)). These lead to an identical read-



(a) Normalized C_{WL}^{MAX} (b) ΔC_x^i variations

Figure 21: Intra-layer C_{max}^{WL} similarity.

Based on our observations, we conclude that DEAR can efficiently predict the per-WL error status by exploiting SAM. Strong intra-layer similarity implies that if we find out the error status from *any* WL on the same h-layer, they can be safely applied to the remaining WLs on the same h-layer without the additional cost. For example, if we measure RBER once for the leading WL on an h-layer, the RBER values for the remaining WLs on the same h-layer can be accurately predicted. Furthermore, when the number of WLs in an h-layer increases (e.g., from 4 to 8 [59, 60]), the efficiency of exploiting intra-layer similarity improves even further by a factor of 4 to 8, as more WLs can be bypassed.

4.3.2 Correlation of Fail-Bit Count for FEM

To validate the feasibility of FEM, we conduct a correlation analysis between $C_{FBC}(ER)$ and the actual error status of WLs. While repeating reads, we track $C_{FBC}(ER)$ and measure each WL's error status under different flash aging conditions. Figure 22 represents the relationship be-

disturb effect across WLs in the same h-layer.

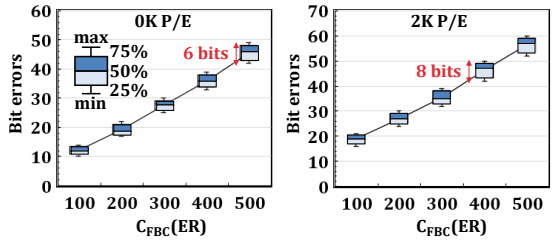


Figure 22: Correlation between C_{FBC} and errors.

tween the $C_{FBC}(ER)$ values and the corresponding measured bit error ranges of WLs, visualized using standard box plots with the 25th and 75th percentiles. The evaluation results show that, across all tested WLs, those with the same $C_{FBC}(ER)$ exhibit tightly clustered error rates. For example, the maximum variation in bit errors among WLs with the same $C_{FBC}(ER)$ is only 8 bits—indicating a strong correlation between $C_{FBC}(ER)$ and read-disturb error. Furthermore, such a strong correlation is sustained under different P/E cycles. Although the maximum bit error variation is slightly larger (from 6 bits to 8 bits, on average) due to inherent flash noise factors (e.g., random telegraph nosie [61] or back-pattern dependency [62]), $C_{FBC}(ER)$ can still be a precise proxy to estimate the read-disturb error of each WL.

To account for the error variation observed in Figure 22, DEAR conservatively maps each C_{FBC} value to the worst-case bit error rate (i.e., the 100th percentile within the observed error range). For instance, if the error threshold for triggering a refresh is E_{TH}, DEAR ensures reliability by initiating RR based on the lowest C_{FBC} value whose worst-case error mapping

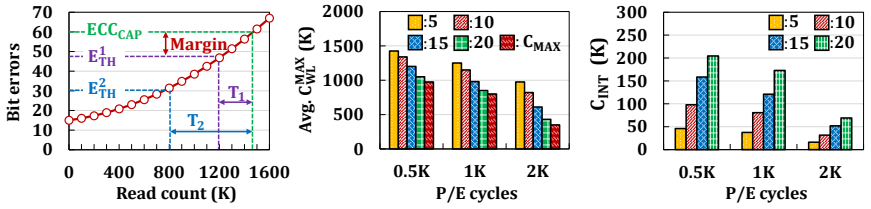
exceeds E_{TH}. We refer to this value as C_{FBC}^{TH}, which is formally defined as:

$$C_{FBC}^{TH} = min\{C_{FBC} \mid M_{RBER}(C_{FBC}) \geq E_{TH}\}, \tag{4.1}$$

where $M_{RBER}(C_{FBC})$ represents the maximum measured number of bit errors associated with a given C_{FBC}. Using Equation 4.1, DEAR determines C_{FBC}^{TH} and performs RR if the C_{FBC} of a WL is greater than or equal to C_{FBC}^{TH}. This approach may cause some WLs to be refreshed before their actual error rate reaches E_{TH}, but this slight underestimation is a reasonable trade-off given the significant monitoring efficiency enabled by FEM.

4.3.3 Monitoring Interval Analysis for BPM

To ensure that all WLs in the monitored block remain reliable throughout the next monitoring interval, the interval C_{INT} must be carefully calibrated based on the selected error threshold E_{TH}. In DEAR, a WL is refreshed when its error rate—approximately monitored via C_{FBC} —reaches E_{TH}. The error threshold E_{TH} defines a fundamental trade-off between refresh frequency and monitoring overhead. A higher E_{TH} allows WLs to tolerate more read operations before triggering RR, thereby reducing the number of RR invocations. However, this comes at the cost of a smaller safety margin, requiring more frequent monitoring. Conversely, a lower E_{TH} provides a larger safety margin, enabling less frequent monitoring, but causes the WLs to be refreshed too earlier. Figure 23(a) illustrates this trade-off and the procedure for determining a reliable C_{INT}. For example, when E_{TH} is set to 80% of ECC_{CAP} (denoted as E_{TH}^{1}), the safety margin is 20%, and C_{INT}



(a) Safety margin (b) RR efficiency (c) Maximum allowable C_{INT}

Figure 23: Determination of monitoring interval C_{INT}.

must be set to less than T_1 to ensure the RBER at the next monitoring point remains below ECC_{CAP}. Alternatively, if E_{TH} is reduced to E^2_{TH}, the safety margin increases, allowing C_{INT} to be extended to T_2. To ensure reliability between monitoring intervals at a given E_{TH}, C_{INT} should be set to the minimum number of reads required for any WL to accumulate errors from E_{TH} to ECC_{CAP} (i.e., the minimum possible T in Fig. 16(a)). This is necessary because reliability varies across WLs, leading to different amounts of accumulated error even under the same number of reads. By setting C_{INT} based on this minimum T across WLs, DEAR strictly guarantees that if the error of the currently monitored WL is below E_{TH}, its error will remain below ECC_{CAP} at the next monitoring point.

To quantitatively understand the impact of E_{TH} on RR efficiency, we evaluate the average C^{WL}_{max} of WLs under different E_{TH} values. We first characterize the corresponding C^{TH}_{FBC} for each E_{TH}, and then measure the maximum number of reads each WL can tolerate before its C_{FBC} reaches C^{TH}_{FBC}. Figure 23(b) shows the average C^{WL}_{max} for four different thresholds: 1) $E_{TH}(5)$, 2) $E_{TH}(10)$, 3) $E_{TH}(15)$, and 4) $E_{TH}(20)$, where $E_{TH}(\mathbf{X})$ indicates a safety margin of $\mathbf{X}$%. For comparison, the conventional C_{max} is also plotted. As

expected, RR efficiency improves significantly as E_{TH} increases. For example, the average C_{max}^{WL} under $E_{TH}(5)$ is 2.8x higher than C_{max} under 2K P/E cycles. In contrast, decreasing E_{TH} reduces RR efficiency, as WLs are refreshed earlier. For instance, $E_{TH}(20)$ yields only 1.2x higher C_{max}^{WL} compared to C_{max}.

Figure 23(c) shows C_{INT} values, characterized using 3,686,400 WLs, across different E_{TH} settings. When the safety margin is set to 5% to maximize RR efficiency, a short monitoring interval of 19.3K reads is required at 2K P/E cycles. In Section 4.4, we demonstrate that DEAR can support such short C_{INT} values with negligible overhead, enabled by SAM and FEM—two optimizations specifically designed to minimize monitoring latency.

4.4 DEARSSD: System Integration

In this section, we present the comprehensive design of DEAR-enabled SSD, called DEARSSD, highlighting its key architectural features and analyzing the associated overheads.

C_{FBC} Measurement. As described in Section 4.3.2, DEAR uses $C_{FBC}(ER)$ as a proxy for error monitoring. To enable low-latency C_{FBC} measurement, DEARSSD leverages two key features of modern NAND flash memory: 1) data randomization, and 2) an on-chip counter, as illustrated in Figure 24. Because all data is randomized prior to programming, each V_{TH} state is uniformly distributed across flash cells in a WL [41, 19, 63] (e.g., 16K cells per V_{TH} state in TLC flash memory; ❶ in Figure 24). As a result, the $C_{FBC}(ER)$ can be efficiently calculated by subtracting the initial ER state count (i.e.,

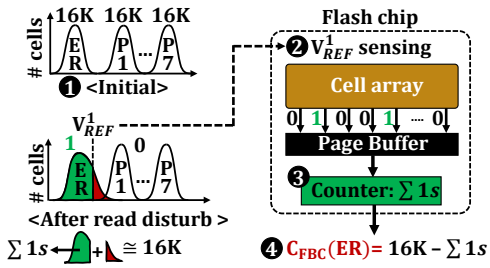


Figure 24: C_{FBC} measurement.

16K) from the number of '1' bits detected after sensing with V^1_{REF} (❷). This bit count is efficiently obtained using a 1s counter embedded within the flash chip, which performs the counting internally and sends only the result to the SSD controller, bypassing ECC engine (❸ and ❹). Both steps can be supported by existing two special flash commands (e.g., SLC sensing and on-chip fail-bit counter [64, 14, 19, 18]), allowing DEARSSD to measure the $C_{FBC}(ER)$ of a WL in just 28 us (20 μs sensing [18] + 8 μs counting [64]).

Error Monitoring Task Scheduling. The error monitoring task is highly optimized in a latency-efficient way in our technique. However, when it conflicts with the user read request, it can cause undesirable delays in response time. For example, monitoring a single 120-MiB block with 321 h-layers takes up to 6.42 ms, even when using SAM and FEM. To address this, DEARSSD performs error monitoring in an interleaved manner. The monitoring task is divided into smaller phases, each responsible for checking a single WL, and it is mixed up with user read requests. For example, in a block with 321 h-layers and a C_{INT} of 19.3K reads, DEARSSD checks only one WL every 60 reads (i.e., $\frac{19.3K}{321}$), a value denoted as C^{WL}_{INT}, instead of monitoring all WLs at once after every C_{INT} reads. Formally, C^{WL}_{INT} is defined

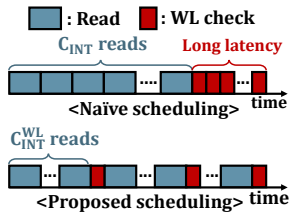


Figure 25: Interleaved monitoring.

as $\frac{C_{INT}}{\text{Number of h-layers}}$.

Figure 25 compares the execution timeline of this interleaved monitoring strategy with the naive approach when a block performs consecutive reads. By distributing the monitoring process over time, DearSSD effectively amortizes the monitoring latency, enabling efficient and scalable error tracking even in large flash blocks.

RR Invocation in DEARSSD. Figure 26 illustrates an operational overview of DEARSSD. Whenever a block is read, DEARSSD checks whether the block's read count is a multiple of C_{INT}^{WL} (❶ in Figure 26). When it reaches such a condition, DEARSSD evaluates the $C_{FBC}(ER)$ of the target WL pointed by per-block WL Pointer (WP) entry (❷ and ❸). WP is a new data structure to indicate which WL should be monitored next. If the measured $C_{FBC}(ER)$ exceeds C_{FBC}^{TH}, the target WL is refreshed along with its neighboring WLs within the same h-layer (❹). The refreshed WLs are migrated to a new block, and the FTL updates the mapping table by invalidating their original locations and remapping the corresponding LBAs to the new locations. To support DEAR, two new parameters—C_{INT}^{WL} and C_{FBC}^{TH} —are required for each P/E cycle range. These parameters are decided offline by real-device characterization and stored in flash memory as a reference table (❺). At run-

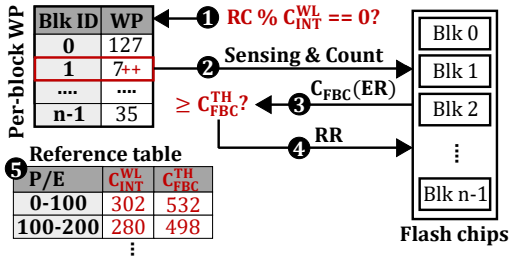


Figure 26: RR invocation flow.

time, DEARSSD loads the reference table into internal memory based on the aging status.

Garbage Collection. DEAR selectively refreshes only the highly disturbed WLs rather than the entire block, which may result in some fragmented blocks. However, these fragmented blocks do not impact SSD functionality, as they are eventually reclaimed through garbage collection (GC) process. Since GC is automatically triggered when the number of available free blocks falls below a threshold and selects victim blocks based on the existing policy, DEAR does not require any modifications to standard GC operations and can be seamlessly integrated into the SSD.

Implementation Overhead. DEARSSD requires only minor changes to conventional SSD. First, DEARSSD leverages existing `GET/SET FEATURE` commands for V^1_{REF} sensing and 1s counting, requiring no hardware modifications to current NAND flash chips. Second, the memory overhead from the newly introduced data structures, the per block WP and the reference table, is also negligible in practice. Each per-block WP entry is 2 bytes, and assuming an 88-MiB block size, DEARSSD requires only about 24-KiB additional space for a 1-TiB SSD. The reference table, which stores

pre-characterized values for each P/E cycle range (e.g., every 100 cycles), requires 8 bytes per entry. With a 2K P/E cycle limit, the total table size is only 160 bytes. The reference table is stored alongside permanent data (e.g., firmware code) in the flash memory, as it contains values characterized offline, and is loaded at runtime to enable the error estimator. Finally, the performance overhead of error monitoring is minimized through latency-efficient design. For example, if C_{INT}^{WL} is set to 60 reads (requiring a total of 2400 μs), only 28 μs are needed for the monitoring task, resulting in less than 1.2% overhead. A detailed runtime overhead analysis is provided in Section 4.5.5.

4.5 Evaluation

4.5.1 Evaluation Methodology

To evaluate the effectiveness of DEAR, we used NVMeVirt [65], a state-of-the-art SSD emulator. We extended NVMeVirt to emulate heterogeneous read-disturb characteristics by modifying its internal NAND flash model. Using our characterized C_{max}^{WL} distributions, we assigned C_{max}^{WL} values—measured at different P/E cycles—to each emulated WL across blocks in NVMeVirt.

Depending on P/E cycles, each emulated WL dynamically reflects realistic read-disturb behavior by referencing its assigned C_{max}^{WL} at runtime. These C_{max}^{WL} values were obtained from our characterization study using error monitoring based on $C_{FBC}(ER)$ under an $E_{TH}(5)$ configuration (i.e., with a 5% safety margin). To precisely emulate read-disturb errors in WLs, we al-

Table 2: Key parameters of a target SSD.

Configuration	480-GiB total capacity; 8 channels; 2 dies/channel; 4 planes/die; 176 layers; 232 blocks/plane; 2112 pages/block; C_{INT}^{WL}: 261 reads (0.5K P/E cycles), 212 reads (1K), 108 reads (2K)
Latencies (μs)	$\texttt{tR} = 40$; $\texttt{tPROG} = 380$; $\texttt{tERASE} = 3500$; $\texttt{tSENSE} = 20$; $\texttt{tCOUNT} = 8$;
Interface	External: PCIe 4.0, 4-lane (8.0 GB/s); Flash channel: Toggle 5.0 (2.0 Gbps)

locate a dedicated read-disturb counter for each WL. At runtime, whenever a block performs a read operation, we increment the read-disturb counters of all other WLs in the block (excluding the currently read WL) by 1. We then determine whether the read-disturb error of each WL has reached E_{TH} by checking if its read-disturb counter has reached its corresponding C_{max}^{WL}. As summarized in Table 2, we configured the architecture and key parameters of the emulated SSDs to be close to modern high-end SSDs employing state-of-the-art 3D flash memory [66, 67, 68].

For our evaluations, we study ten workloads from three benchmark suites, FIO [45], Filebench [69], and Yahoo! Cloud Service Benchmark (YCSB) [70], which are widely used for modern storage system evaluation [71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81]. Table 3 summarizes the I/O characteristics of the workloads used for our evaluations. Each workload uses a dataset of approximately 400 GiB and is run for over 3 hours to ensure steady-state evaluation, allowing us to thoroughly analyze not only RR but also overall impacts such as GC.

We compared four SSD configurations with different read-disturb management techniques: Baseline, Cocktail, DEAR, and DEAR+. Baseline represents an SSD that (i) adopts a priority-aware scheduling algorithm [23]

Table 3: Key I/O characteristics of evaluated workloads.

Workload	Read ratio	Write pattern	Workload	Read ratio	Write pattern
RandRead	1.0	N/A	**YCSB-B**	0.95	Rand.
Web	0.91	Seq.	**YCSB-C**	1.0	N/A
Proxy	0.83	Seq.	**YCSB-D**	0.95	Seq.
Video	1.0	N/A	**YCSB-E**	0.95	Seq.
YCSB-A	0.5	Rand.	**YCSB-F**	0.5	Rand.

to prioritize servicing user I/O requests over RR/GC-induced operations (read/write/erase), and (ii) invokes the RR procedure based on a block-level C_{max} for all blocks. Baseline also adopts the erase suspend/resume feature to reduce the tail latency that occurs when a user read conflicts with a block erase operation. Cocktail builds upon Baseline by incorporating a representative state-of-the-art read-disturb management technique [22]. During RR, Cocktail identifies frequently accessed pages (i.e., read-hot pages) and distributes them across multiple free blocks. This approach reduces the likelihood of future RR invocations by balancing read access patterns across blocks. DEAR is our proposed scheme, which performs WL-level refresh based on runtime error monitoring. We also validate a new optimization, DEAR+, that integrates DEAR and Cocktail. The effectiveness of DEAR+ proves that DEAR can be easily integrated with existing optimization, indicating the high scalability of our proposed technique.

4.5.2 RR-Induced Writes

We first evaluated the effectiveness of DEAR in reducing RR-induced writes. To do this, we measured the internal writes of the SSD for each

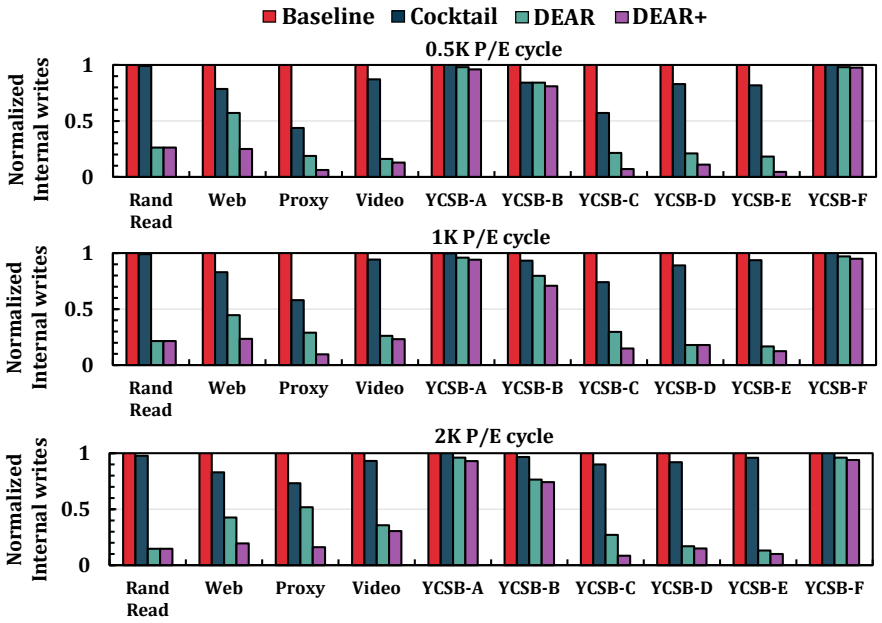


Figure 27: Comparisons on the number of internal writes at different P/E cycles: 0.5K, 1.0K and 2.0K.

workload. These internal writes include not only those caused by RR, but also those from GC-induced writes. Figure 27 compares the internal writes of four SSD configurations, normalized to Baseline, across various P/E cycle counts.

We make four key observations from the results. First, by allowing each WL to tolerate up to its near-optimal C_{max}^{WL}, DEAR significantly reduces RR invocations. Under read-intensive workloads (read ratio ¿ 80%), DEAR achieves up to an 86.8% reduction in RR invocations (64.2 % on average) compared to the Baseline.

Second, Cocktail also reduces RR invocations—by up to 42.8% (with an average reduction of 16.3%)—but its effectiveness is limited compared

to DEAR. Cocktail is particularly effective for workloads with high temporal locality (e.g., YCSB-C), as it distributes read-hot data across blocks. However, its benefit diminishes significantly for workloads with low locality, such as random read-intensive workloads. Moreover, as the SSD ages, C_{max} must be lowered to preserve reliability, which causes the block's read count to exceed C_{max} more frequently—even when read-hot pages are scattered—further reducing efficiency.

Third, DEAR+ achieves additional improvement by leveraging the complementary nature of the two techniques. While Cocktail reduces future RR pressure by balancing access patterns, DEAR relaxes the trigger condition for RR by enabling each WL to tolerate more reads. As a result, DEAR+ reduces RR invocations by up to 95.4% (with an average reduction of 75.6%) over Baseline.

Fourth, DEAR becomes less effective as the write ratio of a workload increases. In read/write mixed workloads such as YCSB-A and YCSB-F, DEAR reduces total writes by only about 3% and 5% over the Baseline, respectively. As the write ratio rises (i.e., fewer reads), read-disturb errors become less frequent, and many blocks are reclaimed by GC before RR is needed. As a result, the impact of read-disturb diminishes. This contrasts with read-intensive workloads, where RR-induced writes outweigh internal writes from GC.

4.5.3 Read Latency

We evaluated the benefit of reduced RR-induced writes on the read tail latency, which is a critical performance factor for many data-intensive

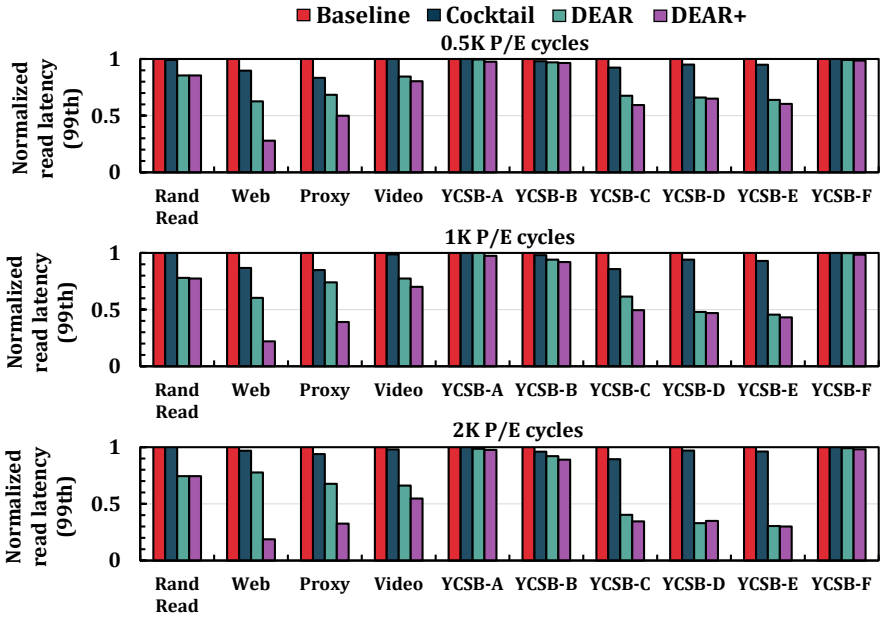


Figure 28: The 99th read tail latency at different P/E cycles: 0.5K, 1.0K and 2.0K.

applications.

Figures 28 and 29 compares the 99th and 99.9th percentile read latencies, τ_{99P} and $\tau_{99.9P}$, in the four emulated SSDs at three different P/Es.

We make two key observations. First, our proposals significantly reduce both τ_{99P} and $\tau_{99.9P}$ compared to Baseline by minimizing additional read/write/erase operations from premature RR invocations. Second, DEAR provides more significant benefits at higher P/E cycles from the standpoint of read performance. For example, at three P/E cycles (e.g., 0.5K, 1.0K, and 2.0K), DEAR reduces τ_{99P} ($\tau_{99.9P}$) over Baseline by 24.4% (46.7%), 29.8% (50.9%), and 35.9% (54.6%) on average, respectively, under read-intensive workloads. This is because, as SSDs age, the frequency of RR invocations

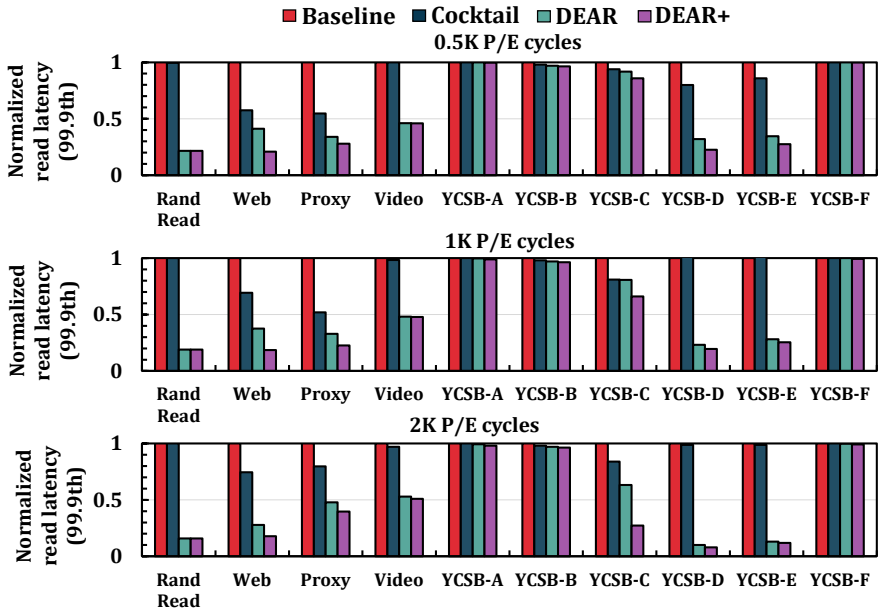


Figure 29: The 99.9th read tail latency at different P/E cycles: 0.5K, 1.0K and 2.0K.

inevitably increases, resulting in more frequent conflicts between user requests and I/O operations from RR procedures. When combined with Cocktail, DEAR+ further reduces τ_{99P} ($\tau_{99.9P}$) by 34.2% (53.4%), 43.8% (57.7%), and 52.3% (62.8%), respectively. Our evaluation results indicate that DEAR can effectively mitigate significant read latency fluctuations caused by frequent RR invocations in modern SSDs.

4.5.4 SSD Lifetime

To evaluate the impact of DEAR on SSD lifetime, we measured the WAF, a key metric that directly influences SSD endurance. Figure 30 presents the WAF of Cocktail and DEAR, normalized to Baseline, across different

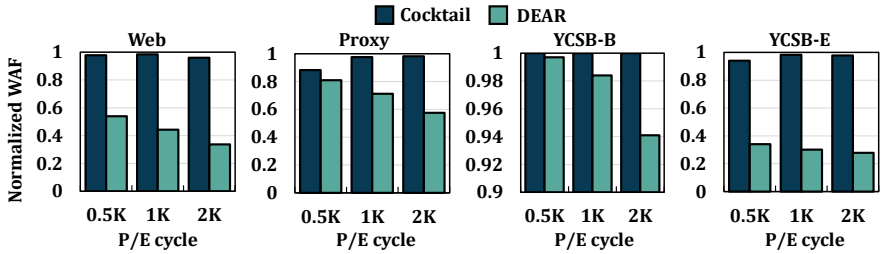


Figure 30: A comparison of WAF improvement.

P/E cycles. In this evaluation, we used four workloads, excluding workloads with read-only I/O patterns.

We make two key observations from the results. First, DEAR significantly improves SSD lifetime by reducing internal writes incurred by RR. For example, DEAR reduces the WAF by up to 72.2% and 70% (46.7% and 44.7% on average) at 2K P/E cycles compared to Baseline and Cocktail, respectively. Second, the benefits of DEAR on SSD lifetime vary depending on the I/O pattern. DEAR achieves substantial WAF reduction in sequential write workloads (e.g., Web, Video), but is less effective for random write workload (e.g., YCSB-B). This is because random write workloads inherently suffer from higher WAF due to poor garbage collection efficiency. Given that modern applications such as AI inference and big data analytics exhibit read-heavy, update-light I/O patterns [82, 83, 84, 85], we believe DEAR will be broadly effective in improving SSD lifetime across a wide range of real-world workloads.

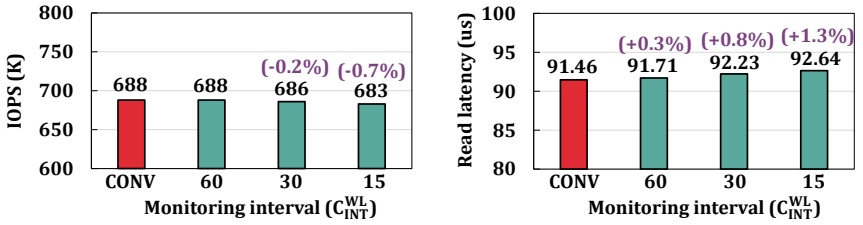


Figure 31: Monitoring overhead analysis.

4.5.5 Monitoring Overhead

As discussed in Section 6, DEAR triggers per-WL error monitoring each time a block performs C_{INT}^{WL} reads. To quantitatively assess the performance impact of this periodic monitoring on normal I/O, we evaluated both IOPS and latency for 4-KiB random reads in the absence of RR invocations. Additionally, to verify the scalability of DEAR, we conducted experiments by reducing C_{INT}^{WL} from 60 reads (corresponding to a 120-MiB block with 321 h-layers) to 15 reads (representing a 480-MiB block with 1284 h-layers). As shown in Figure 31, DEAR effectively amortizes the performance impact of monitoring by leveraging its optimized monitoring techniques. When C_{INT}^{WL} is set to 60 reads, we observe no notable performance degradation compared to conventional SSDs that do not perform error monitoring. Even at an extremely low C_{INT}^{WL} of 15 reads, DEAR introduces only minimal overhead—0.7% drop in IOPS and 1.3% increase in latency. Since DEAR incurs very low overhead even at tight monitoring intervals (e.g., 15 reads), we expect it to be a practical solution for emerging QLC NAND flash, which has an approximately 3× lower C_{max} than TLC, as shown in Figure 12, as well as for increasingly large 3D block sizes.

Chapter 5

Specializing Flash Operations for Efficient In-Flash Processing

Contrary to the common belief that IFP can deliver sufficient performance by executing computation directly inside flash chips, as discussed in Sections 3.1, existing IFP solutions struggle to meet the demanding requirements of data-intensive applications such as LLM inference. At a high level, these limitations stem from two fundamental factors: (1) the high read latency and (2) the high raw bit error rates of NAND flash memory.

In this chapter, we focus on on-device LLM inference, one of the most representative data-intensive workloads. Specifically, Section 5.1.1 first presents an overview of AiF, our proposed IFP scheme, and provides an example illustrating how AiF accelerates LLM inference using IFP. Next, Sections 5.1.2 and 5.1.3 introduce the key techniques of AiF designed to overcome the performance and reliability challenges discussed in Sections 3.1 and 3.3. In addition, Section 5.1.4 and Section 5.2 describe the flash-chip microarchitecture required to support AiF and the system-level requirements for host–AiF co-execution. Finally, Section 5.3 presents comprehensive experimental results that demonstrate the effectiveness of AiF.

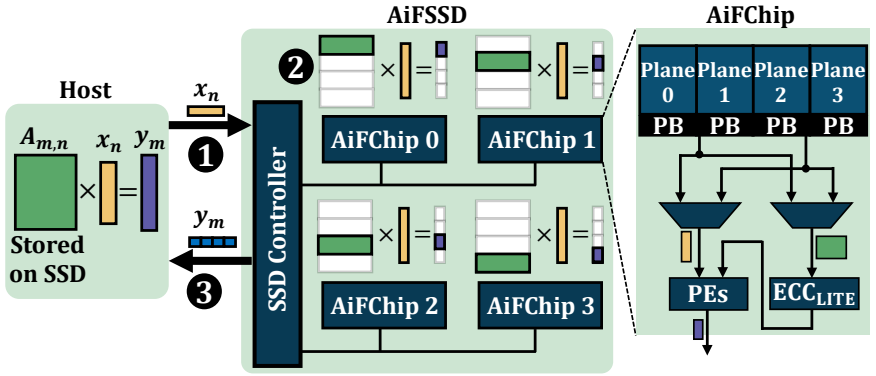


Figure 32: A structural overview of AiF.

5.1 AiF: Accelerator-in-Flash

5.1.1 Overview

We propose AiF, a novel IFP solution specifically designed to optimize in-flash LLM inference. Figure 32 presents a structural overview of AiF. A key component of AiF is the AiFSSD, which is composed of multiple GEMV-enabled flash chips, referred to as AiFChips. (For simplicity, only four chips are depicted.) Unlike baseline SSD offloading schemes that rely on time-consuming I/O to load matrices during inference, AiF performs GEMV directly on the AiFChips where the matrices are stored. Each matrix is divided into equal-sized sub-matrices and stored sequentially across the AiFChips. This enables the AiFSSD to fully utilize the high internal bandwidth of the SSD by operating all AiFChips simultaneously. During GEMV operations, the SSD controller aggregates the sub-vectors from each chip, combines them, and sends the resulting vector back to the host. To support on-chip GEMV operations, each AiFChip is equipped with two components:

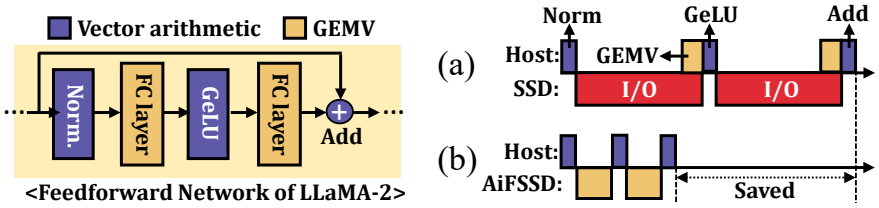


Figure 33: A comparison of (a) the baseline SSD offloading scheme and (b) AiF.

1) ECC_{LITE}, a ligheweight on-chip ECC decoder, and 2) product elements (PEs), consisting of multipliers and an adder tree for vector products.

Figure 33 illustrates an example of the computational process of AiF by comparing its execution timeline with the baseline SSD offloading scheme during the processing of the feed-forward network in LLaMA-2. In the baseline SSD offloading scheme, the matrix must first be loaded from the SSD into memory for GEMV, creating a significant I/O bottleneck. In contrast, AiF utilizes the SSD's high internal bandwidth to perform GEMV, effectively reducing the GEMV computation time. AiFSSD can be integrated with any LLMs by focusing solely on GEMV, the primary bottleneck in on-device LLM inference, while leaving model-specific vector operations (e.g., activation functions) to the host. This decoupling enhances AiF's flexibility across various models.

To address the key limitations of existing IFP solutions (Sections 3.1 and 3.3), which primarily arise from flash read operations, we focus on redesigning the conventional flash read procedure to specifically optimize it for in-flash LLM inference. We introduce two simple yet highly effective techniques, *charge-recycling read* (Section 5.1.2) and *bias-error encoding*

(Section 5.1.3), that significantly enhance both the bandwidth and reliability of parameter reads during inference. These two techniques combined allow AiF to satisfy the high bandwidth demands of on-device LLM inference while maintaining inference accuracy through robust error correction, even with the highly compact on-chip ECC_{LITE}.

5.1.2 Charge-Recycling Read

Key Idea. Cr-read is a technique that specializes the flash read sequence for bulk reads within a block. As described in Figure 3, conventional flash chips include a discharge phase at the end of each read sequence to return to a standby state. This process ensures that a subsequent flash operation can start from the known initial state of block access. However, we observe that a conventional return-to-initial state machine for flash operations may miss optimization opportunities when additional information is available for successive flash operations. Unlike general flash access workloads where subsequent flash operations are difficult to predict, in-flash LLM inference exhibits strong spatial locality when reading model matrices within a flash block. As a result, most voltages applied to WLs and BLs from previous reads can be safely reused without requiring the discharge phase. By leveraging these voltage recycling opportunities inherent in reading successive WLs within a block, cr-read enables AiFChip to read large volumes of LLM parameters much faster.

Figure 34 illustrates how the proposed cr-read scheme works when WL_0 and WL_1 are successively read. In cr-read, after sensing a target WL (② in Figure 34), the WLs and BLs are not fully discharged. Instead, only

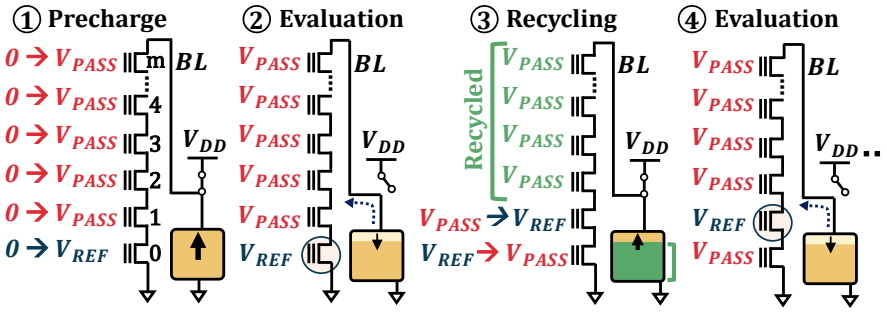


Figure 34: An operational overview of cr-read.

the next read WL discharges from V_{PASS} to V_{REF}, while the previously read WL charges from V_{REF} to V_{PASS}, with all other WLs holding V_{PASS} for the next read (③). Additionally, the BLs are only slightly recharged to compensate for any voltage drop during the previous evaluation phase. Although cr-read can only be applied when successive reads to the same block are requested without any interval, it significantly enhances both read latency and energy efficiency—key design requirements for on-device LLM inference. Figure 35 compares cr-read with conventional reads when reading n WLs consecutively within a block. As shown in Figure 35(a), conventional reads require time-consuming precharge and discharge phases for each WL. The read latency (tR) for each WL in conventional reads can be expressed as:

$$\mathtt{tR} = (\mathtt{tPRE} + \mathtt{tEVAL} + \mathtt{tDISCH}) \tag{5.1}$$

where tPRE, tEVAL, and tDISCH represent the latencies of the prech arge, evaluation, and discharge phases, respectively[1]. In contrast, as shown in Fig-

[1]In this section, we assume that the WLs operate as SLC NAND, requiring only single sensing. This is because applying the bias-error encoding technique, which will be detailed

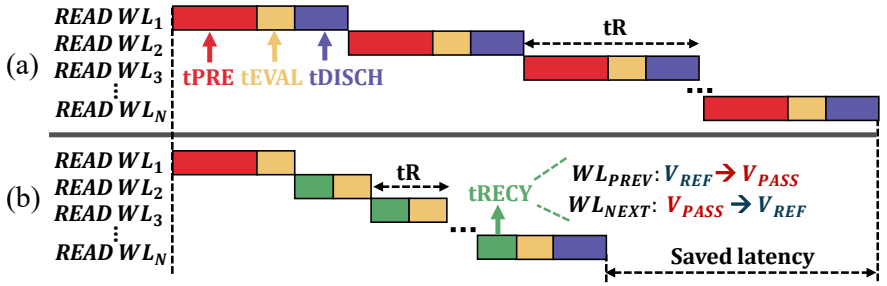


Figure 35: A comparison of (a) conventional read sequence and (b) cr-read.

ure 35(b), cr-read requires only two phases per read: recycling and evaluation (except for the first and last WL reads). Therefore, in cr-read, tR is given by:

$$\texttt{tR} = (\texttt{tRECY} + \texttt{tEVAL}) \tag{5.2}$$

where tRECY denotes the latency of the recycling phase. Moreover, tRECY is considerably shorter than tPRE because tPRE requires charging *all* WLs from 0 V to V_{PASS} (6 V) (i.e., full-range voltage swing), resulting in a high RC delay. In contrast, tRECY only needs to charge *a single* WL from V_{REF} (3.5 V) to V_{PASS} (6 V). The cr-read scheme is effective as well in reducing the energy consumption during read operations. Unlike conventional reads, cr-read only partially charges the previously read WL, raising it from V_{REF} to V_{PASS}. In addition, since the voltage drop across the BLs during evaluation is minimal, only a small amount of current is required to recharge the BLs for sensing the next WL. Consequently, cr-read substantially reduces energy consumption during read operations.

To maximize the bandwidth and energy efficiency of AiFChips during

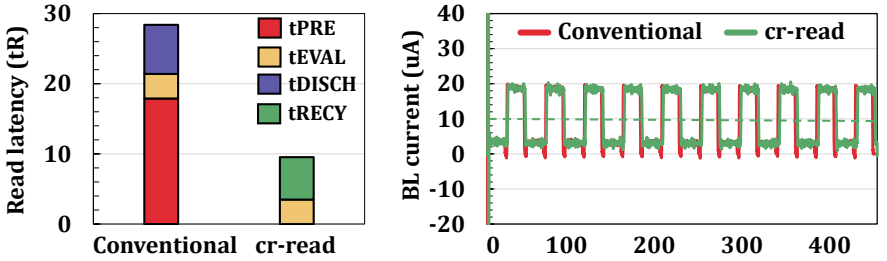
in Section 5.1.3, allows all LLM parameters to be read as if they were SLC.

inference, AiFSSD arranges LLM parameters in a layout optimized for cr-read. Since cr-read achieves peak efficiency when multiple WLs within the same block are read consecutively, AiFSSD organizes each matrix within a same block, allowing for bulk reading of the entire matrix through consecutive cr-read operations. Since LLM parameters have *write once read many* characteristics, AiFSSD can statically pre-arrange each matrix within the same block. Once a matrix is organized within a block, cr-read can be applied for all inference requests to a given model.

Evaluation of cr-read. To validate the effectiveness and practical feasibility of the proposed cr-read scheme, we conducted various characterization studies using both SPICE circuit-level simulations[2] and real NAND flash cell array[3]. To accurately represent the flash cell model in our simulations, we fine-tuned the cell model based on actual measurements from a fabricated real flash cell array. To ensure the accuracy of our simulations, we compared the `tR` and power obtained from our simulator with the measured `tR` (*single sensing read*) of 28 us from the actual product and a reported

[2]Following established methodologies [86, 87, 88], we first conducted measurements on real charge trap flash (CTF) cells, which are widely used for 3D flash memory devices. We focused on the I_d-V_g (drain current vs. gate voltage) characteristics. These provide a foundation for an accurate flash cell model that reflects actual cell behavior. Second, based on the data collected from measurements, we fine-tuned the cell model using the BSIM [89, 90]. At this stage, we verified the model's reliability by comparing the I_d-V_g curves of the tuned BSIM model to those of the real CTF cells, observing an 8.02% difference in drain current, which falls within an acceptable range. Third, circuit modeling. We leveraged the tuned BSIM models and Cadence's Virtuoso [91] to design the NAND flash cell array. The required WL and BL capacitance values were obtained from previous research [92, 93, 94]. Lastly, for simulation, we used Cadence's Spectre [95], an industry-standard SPICE simulator for analog and mixed-signal circuits.

[3]Although simulations offer precise estimations, unexpected real-world factors may impact cr-read 's correct operation on actual flash chips. To ensure the functionality of cr-read, we conducted additional testing on a fabricated CTF cell array chip. This array, a 9 (WLs) by 9 (BLs) cell configuration, allows fine-grained control over each WL and BL, enabling us to validate cr-read 's behavior in a realistic environment.



(a) Improved tR (b) Functionality test

Figure 36: Evaluation results of cr-read.

power value of 24.22 mW [96, 15]. The differences in tR and power consumptions were 2.9% and 0.9%, respectively.

Figure 36(a) compares the read latency (tR) of conventional reads and cr-read. By replacing the time-consuming prech arge and discharge phases with an efficient recycling phase, which introduces only a 6.04 µs delay due to reduced capacitive loads and voltage swing range, cr-read reduces tR by 64%. This substantial reduction in tR allows the AiFChip to achieve an effective bandwidth of 6.4 GB/s when consecutive cr-reads are applied within a block, representing a 2.8x improvement over conventional flash chips.

By minimizing the repetitive precharge of WLs and BLs, cr-read significantly improves the energy efficiency of flash reads. Simulation results indicate that cr-read reduces read energy consumption by 72.1%, from 18.278 pJ/bit to 5.098 pJ/bit.

To validate the real-world functionality of cr-read, we perform experiments using a fabricated CTF cell array. In these experiments, we read the same set of pages using both conventional reads and cr-reads, then compare

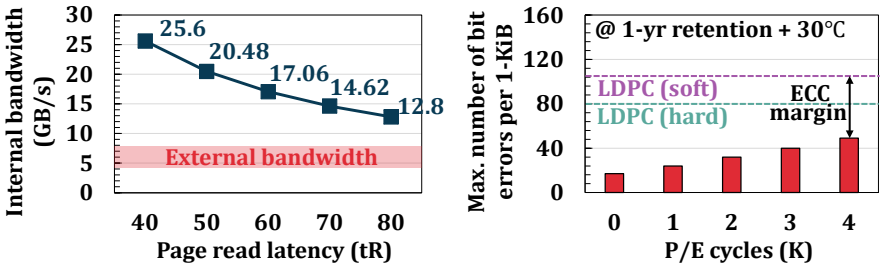
the resulting data to determine whether cr-read introduces any data distortion. Since the tested chip lacks dedicated sense amplifiers, we indirectly assess the data by measuring the current flowing through the BLs during the evaluation phase. As illustrated in Figure 36(b), the measured BL current during cr-read matches that of conventional reads (i.e., produces identical results), confirming that cr-read does not alter the read data and functions correctly.

Feasibility. Another key advantage of cr-read is its ease of implementation. Implementing cr-read requires two main functions: voltage control for each WL and timing control for precharge and discharge phases, both already supported by flash chips. Voltage control can be managed through the X-decoder [97], which selects and drives the appropriate WLs with the required voltages, while timing control can be adjusted by modifying the timer code in the flash scheduler [98]. These adjustments involve only minor modifications to the existing flash chip logic.

5.1.3 Bias-Error Encoding

Key Idea. Be-enc effectively reduces bit error rates in the stored LLM model while slightly sacrificing both read latency and bit error rates for read operations to general pages, i.e., pages that do not store the LLM model. This design is based on two key observations: 1) increased `tR` for general pages has minimal impact on system-level performance, and 2) modern SSDs have a large ECC-capability margin.

Observation 1: `tR` Tolerance. Figure 37(a) shows the internal bandwidth of a 1-TB SSD as read latency (`tR`) is incrementally increased from 40 µs to



(a) Internal bandwidth (b) Large ECC margin

Figure 37: Characteristics of modern SSDs: (a) Variations of internal bandwidth over varying `tR` and (b) Changes in bit errors over increasing P/E cycles.

80 µs. Despite the increase in tR, the internal bandwidth remains constrained by the external bandwidth (e.g., 4-8 GB/s). This indicates that even as `tR` rises (and flash chip bandwidth decreases), the effective user-perceived read bandwidth of the SSD is minimally impacted. From a latency perspective, increases in `tR` also have limited impact. This is because end-to-end latency includes more than just page read latency; it encompasses data traversal through multiple layers, such as the flash channel, PCIe interface, and kernel I/O stack. As a result, slight increases in `tR` can be amortized within the overall latency, exerting limited effect on perceived performance.

Observation 2: Large ECC margin. As previous studies have shown, the error correction capability of the ECC engine employed in modern SSDs significantly surpasses the actual RBER of flash pages, resulting in a large ECC margin [16, 63, 99]. Figure 37(b) shows the maximum number of bit errors per 1-KiB data across various P/E cycles, based on our characterization of 160 real TLC flash chips[4]. We also plot the correction capabili-

[4]We characterized 160 real 3D TLC flash chips using an FPGA-based testing platform

ties for both hard and soft decision decoding in LDPC-based ECC decoders commonly used in SSDs. As shown in the figure, a significant ECC margin exists even under the worst-case operating conditions prescribed by manufacturers (e.g., a 1-year retention time at 4K P/E cycles). This substantial margin stems from two factors: the significant portion of ECC parity reserved in modern SSDs (e.g., 2 KB of parity per 16-KiB data [63]) and the robust error correction capabilities provided by LDPC soft-decision decoding [100, 101, 102].

Based on these observations, the proposed be-enc scheme intelligently manages `tR` and data reliability for the three pages stored within a single WL of TLC NAND flash memory. Be-enc intentionally introduces heterogeneity in the reliability and performance between pages, storing IFP data (i.e., LLM parameters) only on the most reliable and fastest pages. By prioritizing the pages containing IFP data, be-enc effectively increases the SSD's internal read bandwidth during inference while simplifying the on-chip ECC requirements. Be-enc is achieved through the reconfiguration of the V_{TH} state encoding in flash memory. It is important to note that prioritizing the reliability of IFP data over general I/O data ensures that IFP data can be reliably decoded within a flash chip using a compact on-chip ECC decoder. Furthermore, **Observations 1 and 2** (above) indicate that general I/O data does not experience significant degradation in performance or reliability as

equipped with a custom flash controller and a temperature-controlled environment. To ensure comprehensive testing, we selected 120 blocks from each chip, evenly distributed across various physical locations, and evaluated all wordlines (WLs) within each block, totaling 3,686,400 WLs (11,059,200 pages). This extensive testing setup enabled us to obtain statistically significant experimental results. All test procedures follow the JEDEC standards common in the memory industry.

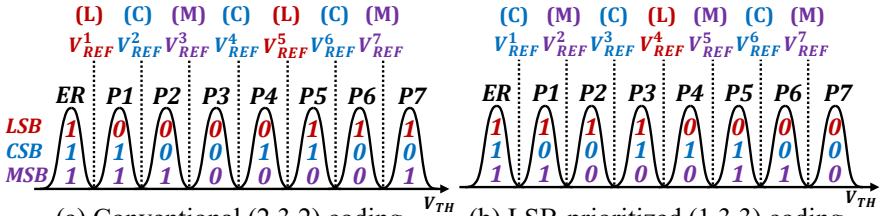
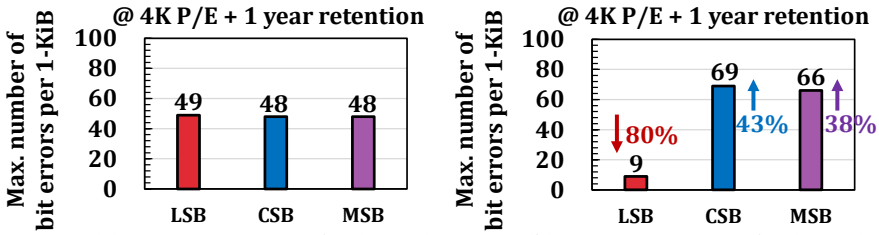


Figure 38: A comparison of (a) conventional V_{TH} state encoding and (b) proposed V_{TH} state encoding.

a result.

Figure 38 illustrates how the prioritized be-enc scheme differs from the conventional V_{TH} state encoding. Figure 38(a) shows the conventional V_{TH} state encoding used in TLC NAND, where (x, y, z) coding represents the number of sensing operations required for reading each page type (i.e., LSB, CSB and MSB). In the (2,3,2) coding, for example, the LSB page requires two sensing operations, while the CSB page requires three. Modern SSDs typically use (2,3,2) coding to balance read time (`tR`) across all pages. Figure 38(b) shows the proposed LSB-prioritized (1,3,3) coding. By reconfiguring the state encoding, the LSB page requires only one sensing operation, making its read performance equivalent to SLC, while the sensing count for the MSB page increases from two to three. Leveraging this intentional `tR` variation, AiFSSD strategically stores IFP data only on the LSB pages. As a result, during inference, all parameters are read solely from the fastest LSB pages.

Another key advantage of be-enc is the enhanced reliability of the LSB pages, which store IFP data. Figures 39(a) and 39(b) present the maximum number of bit errors per 1-KiB data for each page type under (2,3,2) and



(a) Per-page error in (2,3,2) (b) Per-page error in (1,3,3)

Figure 39: Impact of V_{TH} state encoding on page error rates.

(1,3,3) coding, respectively. Compared to (2,3,2) coding, (1,3,3) coding reduces the bit errors of LSB pages by 80%. In (1,3,3) coding, LSB page errors occur only at V^4_{REF}, which distinguishes the P3 and P4 states. Since these states are in the most stable voltage range [103, 104, 81, 63], where fewer V_{TH} shifts occur, the error rates of the LSB pages remain low even under heavy wear (e.g., 4K P/E cycles) and 1-year retention. This significant reduction in errors allows AiFChip to implement a lightweight on-chip ECC.

Feasibility & Overhead. Existing flash chips already support dynamic reconfiguration of the V_{TH} state encoding at runtime [14, 105]. This capability allows be-enc to be applied without requiring any hardware modifications to the flash chip.

A major drawback of be-enc is the increase in the sensing count for MSB pages from 2 to 3, resulting in a longer `tR` for MSB pages. To minimize this overhead, AiFSSD applies be-enc selectively on a per-block basis. Specifically, AiFSSD categorizes flash blocks into two groups: 1) non-IFP blocks, which handle only standard I/O operations and use the conventional (2,3,2) coding, and 2) IFP blocks, which employ (1,3,3) coding

and store both LLM parameters (LSB pages) and general data (CSB/MSB pages). IFP blocks are allocated dynamically from available free blocks whenever LLM parameters need to be stored. By applying be-enc selectively to IFP blocks, only those blocks containing LLM parameters experience performance degradation. We evaluate overhead of be-enc in detail in Section 5.3.2.

5.1.4 Design of AiFChip

This section presents the detailed design of AiFChip, highlighting its key architectural features and analyzing its overhead.

Page Buffer Reuse. As illustrated in Figure 32, each AiFChip performs GEMV operations by multiplying a submatrix, stored sequentially in the IFP block, with an input vector received from the host. This input vector, typically between 8-KiB and 32-KiB, is loaded onto the AiFChip to serve as an operand for GEMV computations.

Instead of requiring a dedicated buffer (e.g., SRAM) for loading the input vector, AiFChip reuses the existing page buffer (PB) allocated to each plane. Originally, the PB functions as a temporary buffer for read, program, and data transfer operations and comprises multiple page latches, each sized to hold a single page. In TLC NAND, each PB includes at least four page latches [106, 107, 3, 14]: one serves as a cache buffer for data transfer, while the remaining three are used as data buffers for page reads. For example, a CSB page read requires three page latches to store data across three sensing operations. However, unlike general I/O operations, AiFChip requires only a single page latch during inference. This is because be-enc enables parameter

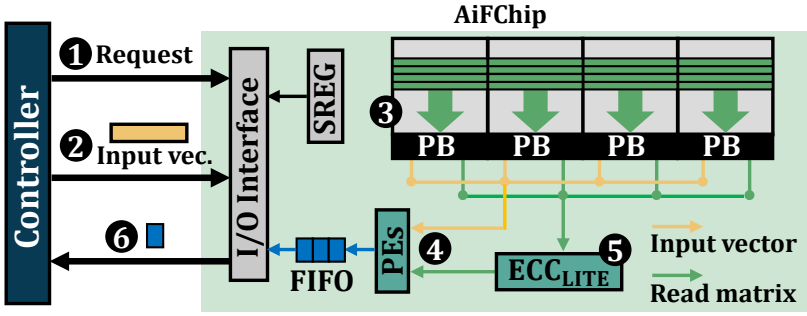


Figure 40: On-chip computation flow.

reads with a single sensing operation, similar to SLC reads. As a result, the remaining two page latches can be repurposed as buffers to hold input vectors, eliminating the need for additional dedicated buffers.

On-Chip Computation Flow. Figure 40 illustrates the overall flow of on-chip GEMV computation. The process begins with the controller sending a command to initiate the GEMV operation (❶ in Figure 40), specifying key parameters: the input vector's dimension, the ID of the block containing the matrix, the offset within the block, and the number of pages to read. The controller then loads the input vector into the PBs (❷), after which planes read matrix sequentially in a continuous flow (❸). As matrix segments are read, the AiFChip retrieves them along with the input vector from the PBs, performing multiplication and accumulation in the PEs (❹). PEs consist of INT8 multipliers and an adder tree to support GEMV operations. Each matrix segment undergoes error correction through ECC_{LITE}, a compact and high-throughput on-chip ECC decoder, before computation (❺). ECC_{LITE} comprises a set of BCH decoders, corrects up to 10-bit errors per 1-KiB data with a 6.4 GB/s throughput. The correction capability of ECC_{LITE} is

Table 4: Breakdown of area and power.

Module Name	Area [mm^2]	Avg. Power [mW]
ECC_{LITE}	0.167	45.1
Processing Elements (PEs)	0.026	3.98
Others	0.016	2.6
Total	0.209	51.68

based on our characterization results in Figure 39(b). After computations, the resulting output vector elements are stored sequentially in the output FIFO. Throughout the computation, the controller polls each AiFChip in a round-robin manner to retrieve results from the FIFO as they are generated (❻), with the FIFO status tracked through the flash chip's status register (SREG).

Area and Power Overhead. For area and power estimation of the AiFChip, we implement the proposed logic in Verilog HDL, synthesizing the designs with Design Compiler at 45 nm technology node and operating frequency of 200 MHz. As detailed in Table 4, the AiFChip's estimated area overhead is 0.209 mm^2, approximately 0.2% of the total flash chip area. The average power consumption is 51.68 mW, with over 87% attributed to ECC. Compared to the baseline ECC decoder analyzed in Figure **??**, ECC_{LITE} requires 15.01× less area and consumes 14.83× less power at the same throughput, enabled by significantly reducing its error correction capability through the use of be-enc.

5.2 Integrating AiF into On-Device LLM Inference

Designing an efficient on-device LLM inference system with AiFSSD requires substantial multi-layer development from application-level software to system software and down to the AiFSSD firmware. In this section, we outline key considerations for building such systems, aiming to provide a broader perspective on AiF.

5.2.1 LLM Inference Workflow with AiFSSD

LLM inference consists of two distinct phases: a prefill phase, which processes the user's input prompt in parallel, and a decode phase, which generates tokens sequentially. In a host system with AiFSSD attached, only the decode phase is offloaded to AiFSSD, while the prefill phase is handled by the host. This phase splitting is based on the consideration that the decode phase requires repeated reads of the entire model for each token generation, making it a key bottleneck in inference [108, 109, 43, 42]. By contrast, the prefill phase requires only a single read of the model and can leverage the host's high computing power to process tokens in parallel.

Figure 41(a) illustrates the computational flow of LLM inference during a decode phase. LLM inference progresses through a series of decoder blocks, where each block takes the output vector (i.e., token embedding) of the previous block as its input. Each decoder block is composed of three main computational stages: QKV generation, multi-head attention (MHA), and feed-forward networks (FFNs). In the QKV generation stage, the in-

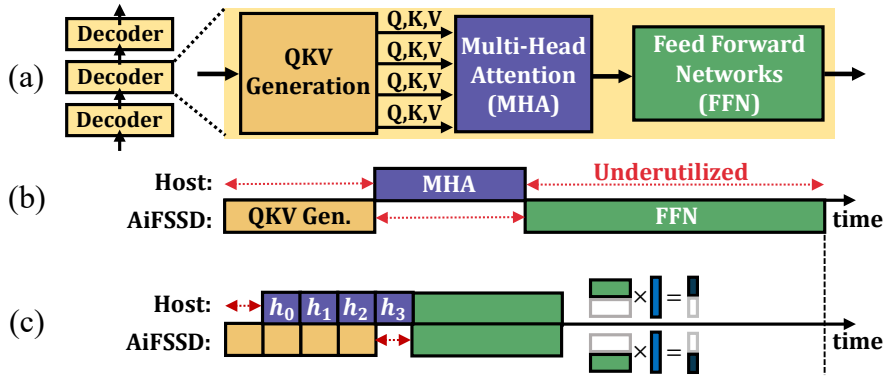


Figure 41: Optimizing LLM inference performance through parallel host-AiFSSD execution: (a) computational flow of LLM inference, (b) a sequential execution flow, and (c) a parallel execution flow.

put vector undergoes linear projections to produce multiple sets of Query (Q), Key (K), and Value (V) vectors. In the MHA stage, each set of Q, K, and V vectors is processed along with previously generated K and V vectors (stored in the KV cache in memory) to produce an attention head. Each attention head is generated independently and then aggregated to form a context-aware token embedding. Finally, in the FFN stage, the token embedding undergoes further refinement through additional linear projections before passing to the next decoder block.

Figure 41(b) illustrates the straightforward scheduling approach for AiF. In this scheme, AiF offloads all stages except the MHA stage to the AiFSSD. The host remains responsible for the MHA stage because it requires access to the KV cache, which must be stored in memory due to its frequent updates.[5] A key limitation of this simple scheduling approach is its sequen-

[5] In on-device LLMs, the KV cache is typically much smaller than model weights due to their low-batch characteristic. Table 6 in Section 5.3.1 details the weight and maximum KV cache sizes (single-batch) of the models evaluated in this work.

tial computation pattern, alternating between the host and AiFSSD instead of engaging them concurrently, resulting in suboptimal performance.

Figure 41(c) illustrates the execution timeline of the proposed parallel scheduling scheme. To enable the overlapped inference with the host, AiF leverages head-level and tensor-level parallelism, inspired by multi-GPU scheduling techniques [110]. As explained above, each attention head (h_i) in the MHA stage can be computed independently with just a single set of Q, K, and V vectors. This independence allows the host to begin processing each head as soon as a vector set is generated, creating an overlap between the Q, K, V generation and MHA stages. Parallel execution is also applied to the FFN stage by dividing the projection matrix into two submatrices, with the host and AiFSSD each processing one independently before aggregating the results. To enable this, the host selectively retains only the necessary portions of the projection matrix in memory during the prefill phase, preparing for parallel processing of the FFN stage in the subsequent decode phase. The number of submatrices that can be loaded (i.e., the degree of tensor-level parallelism) is determined by the available host memory capacity.

5.2.2 System Supports for AiFSSD Integration

This section outlines key design considerations across multiple layers for efficiently integrating AiFSSD.

SSD Extensions. In our design, AiFSSD introduces two specialized commands, (i) `aif_post` and (ii) `aif_gemv`, which extend the existing NVMe protocol. The `aif_post` command stores a matrix in AiFSSD with an optimized layout for IFP. It specifies (1) the matrix dimensions, (2) the host

memory address of the matrix, and (3) the *sequential* LBA range for storing the matrix. Upon receiving `aif_post`, AiFSSD partitions the matrix into equal-sized submatrices and stores them sequentially in IFP blocks across AiFChips (see Figure 17).[6] These blocks are dynamically allocated from the free block list, and AiFSSD maintains a per-block bitmap to differentiate them from non-IFP blocks. The `aif_gemv` command issues a GEMV operation. It specifies (1) the input vector dimension, (2) the input vector's memory address, (3) the matrix's LBA range, and (4) the output vector's memory address. Upon receiving `aif_gemv`, AiFSSD locates the start page addresses of each submatrix and sends the GEMV command, along with the input vector, to AiFChips for computation (steps ❶ and ❷ in Figure 40).[7]

System Software Modifications. In our design, host applications (e.g., an LLM inference engine) communicate directly with AiFSSD over the NVMe raw block interface, bypassing the kernel I/O stack. Although this approach increases application complexity, it provides key benefits. First, it minimizes I/O stack modifications. Only the NVMe driver requires changes. With a userspace NVMe library such as SPDK [113], OS-level modifications, which is often a major barrier to practical deployment, can be avoided entirely. Second, it can maintain optimized data layout. Because the application manually manages IFP data, its sequential placement remains intact,

[6] Each submatrix is initially written sequentially to the LSB pages of the IFP block, while CSB/MSB pages accommodate incoming general data afterward [111, 112]. Although the `aif_post` command ensures sequential placement of submatrices, SSD internal tasks such as garbage collection (GC) may introduce fragmentation. To maintain the optimized layout, AiFSSD performs page copies while preserving the original LSB page order within the IFP block whenever GC is triggered in an IFP block.

[7] Since each submatrix is stored sequentially in IFP blocks, AiFSSD typically needs to look up the mapping table only once per GEMV command. However, if a submatrix spans two IFP blocks, AiFSSD splits the GEMV command into two separate operations.

unaffected by OS services like defragmentation. Third, it reduces communication delays. Bypassing the kernel I/O stack lowers latency, enabling more efficient GEMV execution.

Application-level Requirements. To ensure correct AiF operation, the host application must meet two key requirements: First, it must track matrix LBAs stored in AiFSSD. The application uses the `aif_post` command to store each matrix in AiFSSD. It then records each matrix's LBA range in a *tensor table*, a data structure file that maps tensor identifiers to their corresponding LBAs. During inference, the application consults this table to issue `aif_gemv` with the correct LBA range. Second, it must isolate IFP data. IFP data must reside in a dedicated LBA space, separate from general file system data. This can be achieved by assigning it to a raw partition or a separate NVMe Namespace [114], created using standard Linux tools like fdisk or nvme-cli in Linux.

5.3 System-Level Evaluation

We evaluate the effectiveness of AiF at the system level by integrating a state-of-the-art SSD emulator [115] and an on-device LLM inference engine.

5.3.1 Evaluation Setup

Methodology. We build a virtual AiFSSD using NVMeVirt [115], an open-source SSD emulator. NVMeVirt supports precise timing models of SSD internal operations, such as flash read latency, and enables full-system eval-

uation by mimicking a virtual PCIe endpoint device. Leveraging these capabilities, the virtual AiFSSD accurately models the data flow and timing characteristics of AiFSSD. As an on-device LLM inference engine, we employ llama.cpp [116], a widely adopted on-device engine that offers flexibility and low-level accessibility.

In order to integrate the virtual AiFSSD with llama.cpp, we extend NVMeVirt to support `aif_post` and `aif_gemv` requests, as described in Section 5.2.2. A modified NVMe device driver in the Linux kernel enables llama.cpp to send requests directly to AiFSSD from userspace via the ioctl system call.

During inference, llama.cpp issues GEMV requests to AiFSSD whenever a matrix offloaded to the SSD requires GEMV computation. AiFSSD then computes the GEMV delay using its internal timing model, generates an output vector, and transfers it to the host with an interrupt. For efficient evaluation, we design the virtual AiFSSD to simulate the delay and provide the dummy vector instead of performing the actual computation. Upon receiving the dummy output vector, llama.cpp continues the inference flow as if it were the real GEMV output. Note that the LLM inference flow is deterministic; the contents of the output vector do not affect performance evaluation.

Evaluated Systems. We evaluate the following systems:

• **In-Memory:** The In-Memory scheme represents an ideal system without memory capacity constraints, employing 128-GB of DDR5 memory (up to 86.4 GB/s bandwidth) to store all model parameters in host memory. In-Memory employs an Intel Core i9-14900KS processor as a computation

unit. We exclude dedicated accelerators (e.g., NPUs) for on-device LLM inference, as their low arithmetic intensity does not benefit from accelerators when memory bandwidth is consistent.

• **Memory+SSD:** The Memory+SSD scheme is a baseline scheme that offloads model parameters to a 1-TB SSD due to limited memory capacity. We set the memory capacity to 8-GB. In Memory+SSD, the host memory can store only a portion of the model parameters. Whenever llama.cpp requires model parameters offloaded to the SSD, Memory+SSD fetches them into host memory before computation.

• **AiF:** AiF employs a 1-TB TLC AiFSSD composed of 16 AiFChips along with a host memory capacity of 8-GB. Each AiFChip achieves an effective bandwidth of 6.4 GB/s by employing two proposed flash read optimization techniques: cr-read and be-enc. The maximum bandwidth of AiFSSD for acceleration is 102.4 GB/s (6.4 GB/s $\times$ 16). To maximize generation throughput, AiF leverages head-level and tensor-level parallelism to enable concurrent execution with the host, as described in Section 5.2.1. The degree of tensor-level parallelism is limited by the available host memory capacity.

• **AiF- -:** AiF- - operates identically to AiF, including parallel execution with the host, but does not support cr-read and be-enc techniques. In AiF- -, each flash chip provides an bandwidth of 1.6 GB/s, limiting the maximum bandwidth of AiFSSD to 25.6 GB/s (1.6 GB/s $\times$ 16). Although ECC_{LITE} cannot perform error correction without be-enc, we assume error correction is feasible to focus on evaluating the performance of AiF- -.

Table 5 summarizes the SSD configurations used in our evaluation. We assume a 1-TB SSD with eight flash channels and two chips per channel.

Table 5: Evaluated SSD configurations.

Configuration	8 channels; 2 chips/channel; 4 planes/chip; 16-KiB page
`tR` (2,3,2 coding)	LSB $= 37\mu$s; CSB $= 46\mu$s; MSB $= 37\mu$s;
`tR` (1,3,3 coding)	LSB $= 28\mu$s; CSB $= 46\mu$s; MSB $= 46\mu$s;
`tR` (cr-read)	9.7μs;
Bandwidth (PCIe)	8.0 GB/s external I/O bandwidth (PCIe 4.0, 4-lane);
Bandwidth (ONFI)	2.0 GB/s flash channel I/O bandwidth;

For conventional SSDs with (2,3,2) coding, the `tR` values for LSB, CSB, and MSB pages are 37 µs, 46 µs, and 37 µs, respectively. For IFP blocks with (1,3,3) coding (i.e., with be-enc applied), the `tR` values are 28 µs, 46 µs, and 46 µs, respectively. The `tR` values are obtained from measurements on a real TLC flash chip. When cr-read is applied, the `tR` value for LSB pages within IFP blocks is reduced to 9.7 µs, based on evaluation results in Section 5.1.2. The external PCIe bandwidth is set to 8.0 GB/s, while each flash channel provides a bandwidth of 2.0 GB/s.

Evaluated Models. We evaluate eight distinct LLMs with various parameter sizes and architectures. Table 6 summarizes their key parameters and architectural details. All models are in INT8 format, the most commonly used format for on-device LLMs. Among these, Mixtral-8x7B adopts a mixture-of-experts (MoE) architecture, a variant of the transformer architecture. Unlike conventional transformers, which access the entire parameter set for each token generation, Mixtral-8x7B dynamically selects a subset of parameters at each step. As a result, while Mixtral-8x7B has a total weight size of 46.2-GiB, it requires reading only 14.7-GiB of parameters per token generation.

Table 6: Evaluated LLM configurations.

Model	Weight (GiB)	KV Cache (GiB)	# Layers	# Heads
LLaMA2-7B [117]	6.7	1	32	32
LLaMA3-8B [118]	8	0.5	32	32
Falcon-11B [119]	11	0.25	40	32
LLaMA2-13B [120]	12.9	1.1	40	40
Mixtral-8x7B [121]	14.7 / 46.2	2	32	32
GPT-NeoX-20B [122]	20.4	1.05	44	64
Falcon-40B [123]	41.4	0.1	60	128
LLaMA3-70B [124]	69.8	1.3	80	80

5.3.2 Throughput

Figure 42 presents the throughput comparison results across 8 different LLMs, highlighting four key observations. First, AiF effectively mitigates the performance bottleneck of the baseline SSD offloading scheme. The Memory+SSD scheme operates efficiently for smaller models where parameters fit in memory (Figure 42(a) and (b)). However, as the model size grows (e.g., Figure 42(c)), a larger proportion of parameters must be offloaded to the SSD, leading to a sharp decline in throughput. In contrast, AiF sustains high throughput for larger models by leveraging the SSD's high internal bandwidth. On average, AiF achieves 14.6x higher throughput than Memory+SSD across the evaluated models.

Second, AiF also outperforms in-memory inference. Despite being limited to 8-GB of memory, AiF achieves an average of 1.4x higher throughput than the In-Memory scheme. This is due to the AiFSSD's high internal bandwidth of 102.4 GB/s , enhanced by cr-read and be-enc, which exceeds the memory bandwidth of 86.5 GB/s. Furthermore, AiF enables concurrent inference between the host and the AiFSSD, allowing the host memory band-

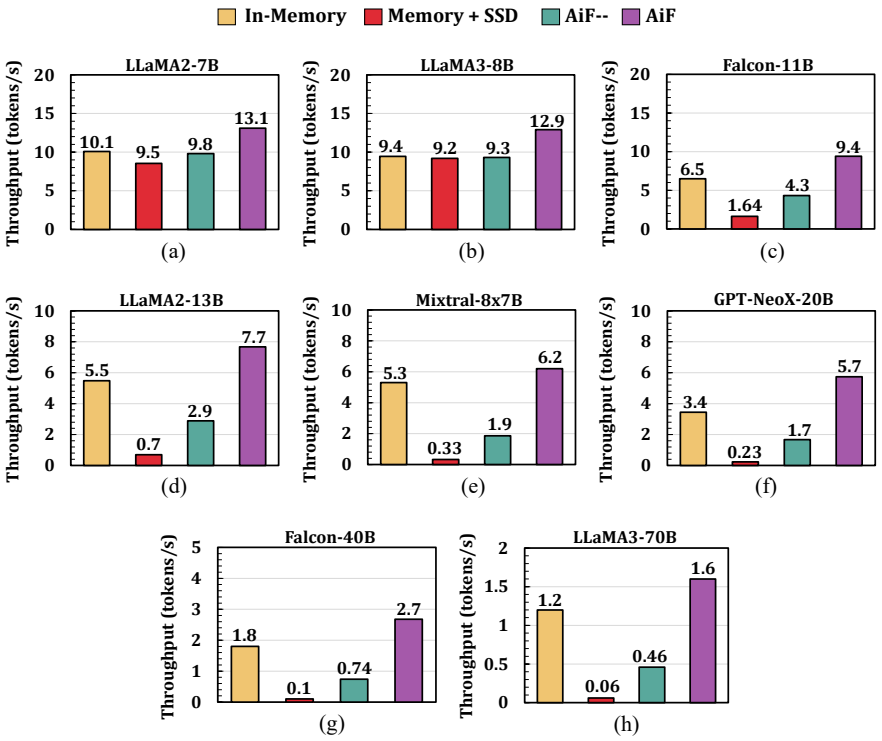


Figure 42: Throughput comparison of In-Memory, Memory+SSD, AiF- -, and AiF.

width and the internal bandwidth of the AiFSSD to be utilized simultaneously. This advantage becomes more pronounced for smaller models, as a larger portion of the submatrix can fit in host memory, increasing the opportunities for concurrent inference.

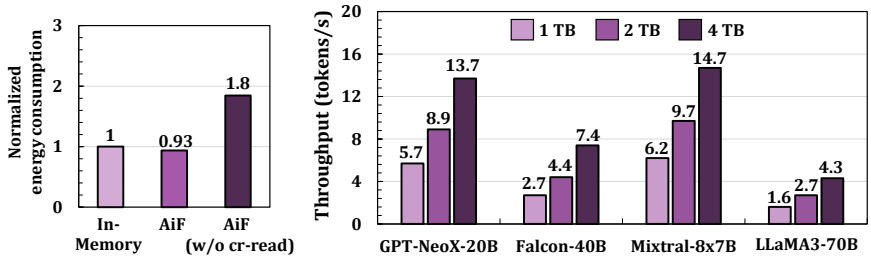
Third, in-flash LLM inference without bandwidth optimization has limited scalability. The AiF- - scheme achieves 4.59x higher throughput than Memory+SSD on average by utilizing the internal bandwidth. However, its performance significantly declines for larger models, achieving only 0.74 tokens per second for a Falcon-40B model. In contrast, AiF sustains 2.7 to-

kens/s for the same model. AiF offers an average of 2.67x higher throughput than AiF- -.

Fourth, MoE architecture aligns well with AiF. To fully leverage the benefits of MoE, both high storage capacity for large model parameters and sufficient bandwidth to access them quickly are essential. In-memory systems provide high bandwidth but are limited by their capacity, while SSDs offer large storage capacity but are constrained by low bandwidth. In contrast, AiFSSDs are particularly well-suited for MoE, as they combine high storage capacity with substantial internal bandwidth. Although AiF performance gradually declines as model size increases (e.g., Figures 42(h)), MoE enhances AiF's scalability, enabling support for larger models while maintaining the same SSD capacity (Figure 42(e)).

5.3.3 Energy Efficiency

Next, we evaluate the energy efficiency of AiF and In-Memory. To understand the impact of cr-read, we also evaluate the energy consumption of AiF without cr-read applied. For the energy model, In-Memory assumes computations are performed on an energy-efficient NPU (1.4 TOPS/W as reported in [125]), and the host's memory is modeled as LPDDR5 (7 pJ/bit based on [126]). Flash reads are modeled using the energy evaluation results from Figure 36, while ECC and GEMV operations are based on the results in Table 4. Flash channel and PCIe energy consumption are assumed to be 5.8 pJ/bit [127] and 7.5 pJ/bit [128], respectively. Figure 43(a) shows the energy consumption for generating 512 output tokens using the Falcon-40B model, normalized to the energy consumption of In-Memory. The results show that



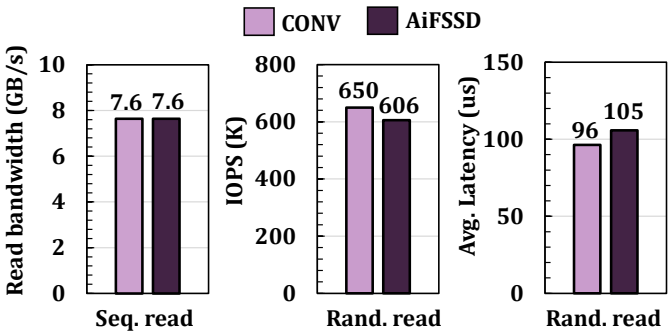
(a) Energy efficiency (b) Scalability of AiF

Figure 43: (a) Comparisons of energy efficiency. (b) Throughput scalability in AiF.

AiF achieves nearly a 2x improvement in energy efficiency when cr-read is applied. This improvement is attributed to parameter reads being the primary energy bottleneck in LLM inference, with cr-read reducing flash read energy consumption by more than 3x. Additionally, AiF and In-Memory exhibit comparable energy consumption, with AiF consuming approximately 7% less energy. This indicates that AiF is a more energy-efficient solution, particularly when considering the high static energy consumption of DRAM as its capacity increases.

5.3.4 Scalability

Figure 43(b) presents the throughput evaluation results for AiFSSD capacities of 1-TB, 2-TB, and 4-TB. While doubling the capacity theoretically doubles the internal bandwidth, the observed performance improvement scales by 1.35x to 1.68x. This sublinear scalability arises from two key factors: First, as shown in Figure 33, LLM inference involves multiple vector arithmetic operations interleaved with GEMV blocks. Consequently, reducing GEMV time alone does not result in a proportional reduction in



(a) Bandwidth (b) IOPS (c) Latency

Figure 44: Overhead analysis.

overall inference time. Second, GEMV requests are handled through the NVMe protocol, which introduces unavoidable control overheads (e.g., interrupts, DMA) that hinder full utilization of internal bandwidth. Furthermore, as the number of flash chips increases with capacity, the control overhead grows, mainly due to the longer time required to load input vectors into flash chips via flash channels. To enhance AiF's scalability, various scheduling optimizations on both the host and SSD sides are necessary. These further enhancements are left for future work.

5.3.5 Overhead Analysis

To analyze the overhead of be-enc, we compare the read performance of a conventional SSD (CONV) with (2,3,2) coding and the AiFSSD with (1,3,3) coding. For a worst-case analysis, we assume all blocks on the AiFSSD are IFP blocks. We evaluate three key metrics for read performance: sequential read bandwidth, random read IOPS, and latency. Figure 44 presents

the evaluation results, leading to two key observations. First, sequential read bandwidth remains stable despite the increased tR. As shown in Figure 44(a), both CONV and AiFSSD maintain a consistent bandwidth of 7.6 GB/s, due to the small external bandwidth relative to the large internal bandwidth of the SSD. Second, random reads exhibit slight performance degradation. In random read workloads, as shown in Figures 44(b) and (c), AiFSSD experiences a 6.8% reduction in IOPS and a 9.3% increase in latency compared to CONV. For random reads, only a fraction of the read page (4 KB out of 16 KB) is delivered to the host, alleviating the external bandwidth bottleneck and exposing the impact of the increased tR. These suggest that if AiFSSD contains a large number of IFP blocks, it may result in some performance loss in random read-intensive applications. We consider this a reasonable trade-off for the significant performance and reliability benefits be-enc provides for IFP.

5.3.6 RR-Induced Writes

Finally, we evaluate the amount of internal writes induced by read refresh (RR) when running AiF. As discussed in Section 3.2, IFP inherently experiences higher read-disturbance stress because it performs a large volume of reads and exploits the high internal read bandwidth of flash chips. This elevated stress leads to more frequent RR invocations, which in turn generate substantial internal data migration and writes. To quantify this effect and evaluate the benefit of integrating DEAR, we compare two schemes. The first is AiF, which employs a conventional block-level RR scheme based on static read-count thresholds. The second is AiF+, which integrates AiF

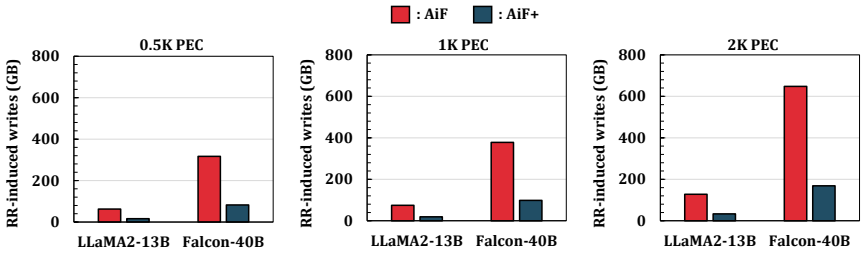


Figure 45: Comparison of RR-induced internal writes for AiF and AiF+ across different P/E cycles (0.5K, 1K, and 2K) over 5000 generated tokens for LLaMA2-13B and Falcon-40B.

with DEAR's dynamic, error-based RR scheme. We evaluate both schemes using LLaMA2-13B and Falcon-40B. During inference, these models perform sequential reads of 6.5 GB and 33 GB of weights per generated token, respectively.

Figure 45 reports the total amount of RR-induced internal writes across different P/E cycles (PECs) while each model generates 5000 tokens. We make three key observations. First, AiF+ significantly reduces RR-induced writes. Since DEAR and AiF address orthogonal aspects of IFP, they can be seamlessly integrated, allowing AiF+ to effectively mitigate the high RR overhead observed in IFP workloads. Across all evaluated settings, AiF+ reduces RR-induced writes by 79.7% on average (up to 86%) compared to AiF.

Second, RR-induced writes increase with model size. For example, when generating 5,000 tokens, Falcon-40B produces approximately 5.07x more internal writes than LLaMA2-13B. This trend is expected because larger models require reading more weights per token, resulting in substantially higher read volume and thus stronger read-disturbance stress. Specifi-

cally, over 5000 tokens, LLaMA2-13B reads approximately 31.73 TB, whereas Falcon-40B reads approximately 161.1 TB, leading to much higher disturbance accumulation.

Third, RR-induced writes remain non-negligible even with DEAR integration under intensive inference. Although AiF+ substantially reduces RR overhead, it still generates approximately 168.5 GB of internal writes at 2K PEC when running Falcon-40B inference. While this overhead may be acceptable for occasional inference, it becomes increasingly problematic for intensive usage, particularly when generating long outputs with many tokens. These results indicate that, even with DEAR, RR-induced writes can still impose non-negligible lifetime overheads, suggesting that further reductions in RR overhead remain an important open challenge. We discuss additional approaches for further reducing RR overhead in IFP as part of our future work (Section 6.2).

Chapter 6

Conclusions

6.1 Summary

To meet the growing performance and capacity demands of modern data-intensive applications, IFP has been actively investigated in recent years. As a new paradigm of computational SSDs, IFP executes application-level computations directly inside flash chips, eliminating excessive data movement between storage and external compute units.

While IFP has the potential to deliver substantial performance benefits by fully exploiting the internal bandwidth of SSDs, several fundamental challenges must be addressed for its practical adoption. These challenges include limited reliability that leads to reduced device lifetime and increased implementation overhead, as well as diminishing performance gains as flash memory capacity continues to scale.

In this dissertation, we proposed a set of software-, microarchitecture-, and circuit-level techniques to effectively address the reliability and performance challenges of IFP.

First, we introduced DEAR, a novel read-refresh scheme that mitigates the lifetime and performance overhead caused by read disturbance, one of the most critical reliability issues in modern SSDs, particularly under read-intensive workloads. We observed that read-disturbance characteristics in

modern 3D NAND flash memory vary significantly across wordlines, rendering conventional block-level, static read-count-based refresh schemes highly inefficient. To overcome this limitation, DEAR selectively refreshes individual wordlines based on their actual error rates rather than block-level read counts. By refreshing only highly disturbed wordlines, DEAR significantly reduces unnecessary data copying during read-refresh operations, thereby improving both SSD lifetime and performance.

Second, we presented AiF, an IFP-specialized SSD architecture designed to accelerate on-device LLM inference. To improve flash read performance, AiF introduces cr-read, which reuses the read voltages applied in previous accesses to enable high-bandwidth reading of bulk data within a block. In addition, AiF employs be-enc, which intentionally creates non-uniform reliability across pages within a block to generate highly reliable pages. By selectively storing model weights only in these reliable pages, AiF significantly reduces the complexity of on-chip ECC decoding while maintaining inference accuracy. By combining these two techniques, AiF achieves substantial inference performance improvements with minimal overhead, effectively addressing the limitations of existing IFP solutions.

Overall, this dissertation demonstrates that carefully co-designed reliability and performance optimizations across multiple system layers are essential for realizing the full potential of in-flash processing. We believe that the techniques proposed in this work provide a solid foundation for the practical deployment of IFP and open new opportunities for accelerating a wide range of future data-intensive applications.

6.2 Future Work

6.2.1 Reliability-Aware Data Placement

As analyzed in Section 4.1, modern 3D NAND flash memory exhibits severe inter-block reliability variation, resulting in a wide dispersion in the maximum tolerable read count before refresh, denoted as C_{max}, across different blocks. This observation suggests that the RR overhead of IFP can be further reduced by reliability-aware data placement. In particular, if read-intensive IFP data (e.g., LLM weights) can be selectively stored in highly reliable blocks with large C_{max}, RR invocations can be substantially decreased. Realizing this approach requires (1) proactively identifying blocks with high reliability and (2) extending the flash translation layer (FTL) to support reliability-aware placement of IFP data. A straightforward way to identify reliable "good blocks" is to measure read-disturbance errors accumulated under a given number of reads and track blocks that exhibit relatively low error growth. By maintaining a pool of such good blocks and preferentially placing read-intensive IFP data into them during writes, the SSD can effectively reduce the frequency of RR and mitigate the associated lifetime and performance overheads.

6.2.2 Accelerating Vector ANNS Using In-Flash Processing

The proposed AiF scheme can be naturally extended to support a broader range of data-intensive applications beyond LLM inference. One promising example is vector approximate nearest neighbor search (ANNS), which

serves as a core operation in emerging workloads such as retrieval-augmented generation (RAG). Vector ANNS requires searching for vectors that are close to a given query vector within massive datasets. Instead of transferring candidate vectors from SSDs to external compute units, IFP can leverage the high internal bandwidth of flash chips to perform distance computations directly inside the storage device. By filtering and forwarding only the nearest candidate vectors to the host, IFP can significantly reduce data movement between the SSD and the host system. One key challenge in supporting vector ANNS with in-flash processing is the high implementation overhead of on-chip ECC, similar to the case of LLM inference. To mitigate this overhead, we plan to explore a two-step ECC decoding strategy that splits the decoding process between the flash chip and the SSD controller. For example, BCH-code-based decoding typically consists of two major stages: syndrome generation and error correction, where the latter stage incurs substantially higher hardware complexity. Based on this observation, the flash chip could perform only the lightweight syndrome-generation stage, together with in-flash distance computation, and then forward both the computed results and the generated syndromes to the SSD controller. The controller can subsequently complete the error-correction stage using the syndromes and correct any error-induced deviation in the distance results. This split-decoding approach has the potential to significantly reduce the on-chip ECC overhead while still ensuring correctness for error-sensitive vector ANNS computations.

6.2.3 Composable In-Storage Processing Framework

Despite numerous studies on ISP aimed at accelerating data-intensive applications using computational SSDs, the practical deployment of ISP in real-world systems remains highly challenging for two primary reasons. First, existing ISP approaches often require application-specific customization of commercial SSD controller chips. However, modern SSD controllers are highly complex, monolithic SoC designs that integrate host interfaces, NAND flash control logic, error correction engines, and firmware. Since these controllers are proprietary technologies developed by storage manufacturers, application-specific customization is largely impractical from an engineering perspective. Second, the performance benefits of current ISP architectures are inherently limited, as computation remains constrained by the bandwidth and processing capabilities of the SSD controller.

To overcome these challenges, we plan to investigate a composable ISP architecture that introduces an intermediate buffer chip with computational capabilities between the SSD controller and flash chips. Inspired by microservice architectures and chiplet-based system designs, this approach aims to provide both flexibility and scalability. Because the buffer chip is decoupled from both the SSD controller and flash chips, it can be developed independently of storage vendors, significantly improving accessibility and deployability. Furthermore, by aggregating multiple flash channels through the buffer chip, the proposed architecture can achieve substantially higher effective bandwidth than existing ISP designs, enabling more scalable and efficient in-storage acceleration.

초 록

낸드 플래시 기반의 SSD는 현대의 데이터 집약적 응용을 위한 주요 저장 매체로 자리 잡았다. SSD는 기존 HDD에 비해 짧은 지연 시간, 소형 폼팩터, 낮은 에너지 소모 등의 다양한 장점을 제공한다. 또한 멀티 레벨 셀과 3D 워드라인 적층 기술의 발전으로 SSD의 저장 용량은 크게 증가하여, 단일 장치에서 수백 테라바이트의 데이터를 저장할 수 있게 되었다. 이러한 장점들과 비약적인 용량 향상에 힘입어 SSD는 현대 데이터 집약적 워크로드를 위한 사실상의 표준 저장 장치로 자리매김하였다.

SSD는 HDD에 비해 현저히 높은 성능을 제공하지만, 제한적인 대역폭으로 인해 데이터 집약적 응용이 요구하는 성능 수준을 충족하는 데에는 여전히 한계가 존재한다. 현대의 데이터 집약적 워크로드는 방대한 데이터셋에 대한 전례 없는 수준의 높은 데이터 처리 성능을 요구한다. 이를 충족하기 위해 현대 컴퓨팅 시스템은 초당 수백 테라플롭스의 연산 성능을 제공하는 고성능 연산 장치(예: GPU)를 활용한다. 그러나 실제로 데이터를 저장하고 있는 SSD의 대역폭은 물리적으로 약 4–15 GB/s 수준에 머물러 있다. 이와 같은 심각한 SSD 대역폭 제약은 연산 성능 향상만으로는 해결할 수 없는 중대한 성능 병목을 초래한다.

SSD의 제한적인 대역폭 문제를 극복하기 위해, SSD 내부의 플래시 칩에서 직접 연산을 수행하는 인-플래시 프로세싱(In-Flash Processing, IFP)이 활발히 연구되고 있다. SSD 내부에는 다수의 플래시 칩이 병렬로 구성되어 있으며, 이들 칩 간의 집적 내부 대역폭은 외부 PCIe 인터페이스 대역폭보다 크다. 따라서 IFP는 SSD의 가용 대역폭을 효과적으로 증가시킬 수 있다. 이러한 장점을 활용하여, 다양한 데이터 집약적 응용을 가속하기 위한 IFP 기반 기법들이 제안되어 왔다.

IFP는 상당한 성능 향상을 제공할 잠재력이 있으나, 실제 시스템에 적용되기 위해서는 몇 가지 핵심적인 문제를 해결해야 한다. 첫째, 높은 대역폭을 확보하기 위해 모든 플래시 칩을 동시에 동작시키는 경우, 심각한 읽기 교란(read disturbance)이 발생하여 SSD의 수명과 성능이 크게 저하된다. 둘째, 저장 용량 중심으로 발전해 온 최신 낸드 플래시 메모리의 설계 추세로 인해 칩당 읽기 대

역폭이 크게 감소하였으며, 이는 IFP가 제공하고자 하는 성능 이점을 직접적으로 약화시킨다. 셋째, 플래시 칩은 매우 엄격한 공정 자원 제약 하에서 제조되므로, 추가적인 IFP 로직을 통합하는 것은 기술적 · 비용적 측면에서 큰 부담이 된다. 특히 신뢰성 있는 IFP 동작을 위해서는 온칩 오류 정정 코드(Error Correction Code, ECC) 디코더가 필수적이며, 이는 상당한 면적 및 전력 오버헤드를 초래한다.

본 논문에서는 IFP의 실용적 적용을 가로막는 이러한 핵심 문제들을 효과적으로 해결하기 위한 두 가지 새로운 기법을 제안한다. 첫째, 읽기 교란의 영향을 완화하기 위해 보다 정밀한 단위로 리프레시를 수행하는 동적 읽기-리프레시 기법인 DEAR(Dynamic Error-Aware Refresh)를 제안한다. 기존 SSD 펌웨어는 블록의 읽기 횟수가 오프라인에서 설정된 읽기 횟수 임계값을 초과하면 리프레시를 수행함으로써 읽기 교란에 대응한다. 그러나 최신 3D 낸드 플래시 메모리는 공정 변이가 매우 크기 때문에, 이러한 단순한 리프레시 정책은 과도하고 불필요한 데이터 마이그레이션을 유발한다. 이를 해결하기 위해 DEAR는 읽기 횟수가 아닌 실제 오류율에 기반하여 리프레시를 수행하며, 블록 단위가 아닌 워드라인 단위로 동작한다. 읽기 교란이 심각한 워드라인만을 선택적으로 리프레시함으로써, DEAR는 리프레시 과정에서 발생하는 데이터 이동을 대폭 감소시키고, 읽기 집약적 워크로드에서 SSD의 수명과 성능을 크게 향상시킨다.

둘째, IFP의 대역폭과 구현 오버헤드를 효과적으로 개선하기 위한 IFP 프레임워크인 AiF(Accelerator-in-Flash)를 제안한다. 플래시 칩의 읽기 대역폭 감소로 인해 제한되는 IFP 성능을 극복하기 위해, AiF는 이전 읽기 연산에서 사용된 전압을 이후 읽기에 재사용하는 전하 재활용 읽기(charge-recycling read) 기법을 적용하여 읽기 대역폭을 획기적으로 향상시킨다. 또한 낸드 플래시 메모리의 높은 오류율을 보정하기 위해 요구되는 온칩 ECC 디코더의 면적 및 전력 오버헤드를 줄이기 위해, AiF는 의도적으로 페이지 간 비균일한 오류 특성을 유도하는 편향 오류 인코딩(biased error encoding)을 채택한다. 이를 통해 매우 높은 신뢰성을 갖는 페이지를 생성하고, LLM 가중치와 같은 IFP 데이터들을 이러한 신뢰성 높은 페이지에 선택적으로 저장함으로써 경량 ECC만으로도 IFP의 신뢰성을 보장한다. 이 두 가지 기법을 결합함으로써 AiF는 기존 IFP 대비 최대 4배 높은 읽기

대역폭을 달성함과 동시에 구현 오버헤드를 크게 개선한다.

제안한 기법들의 효과를 검증하기 위해, 실제 워크로드와 LLM을 대상으로 SSD 에뮬레이션 플랫폼에서 광범위한 실험을 수행하였다. 실험 결과, DEAR는 읽기-리프레시로 인한 데이터 복사를 현저히 감소시켜 읽기 집약적 워크로드에서 SSD의 수명과 성능을 크게 향상시킴을 확인하였다. 또한 AiF의 IFP 특화 읽기 및 신뢰성 기법은 미미한 구현 오버헤드로 읽기 대역폭을 대폭 향상시켜, 온디바이스 LLM 추론 처리량을 크게 개선함을 보였다.

키워드: 인플래시 프로세싱, 낸드 플래시 메모리, 저장장치 시스템, 대형 언어 모델 추론, 성능 최적화, 수명 최적화

학번: 2020-20678

Bibliography

[1] Y. Wang, X. Pan, Y. An, J. Zhang, and G. Reinman, "Beacon-gnn: Large-scale gnn acceleration with out-of-order streaming in-storage computing," in *2024 IEEE International Symposium on High-Performance Computer Architecture*, pp. 330–344, 2024.

[2] Y. Wang, S. Li, Q. Zheng, L. Song, Z. Li, A. Chang, H. H. Li, and Y. Chen, "Ndsearch: Accelerating graph-traversal-based approximate nearest neighbor search through near data processing," 2024.

[3] H.-W. Hu, W.-C. Wang, Y.-H. Chang, Y.-C. Lee, B.-R. Lin, H.-M. Wang, Y.-P. Lin, Y.-M. Huang, C.-Y. Lee, T.-H. Su, C.-C. Hsieh, C.-M. Hu, Y.-T. Lai, C.-K. Chen, H.-S. Chen, H.-P. Li, T.-W. Kuo, M.-F. Chang, K.-C. Wang, C.-H. Hung, and C.-Y. Lu, "Ice: An intelligent cognition engine with 3d nand-based in-memory computing for vector similarity search acceleration," in *2022 55th IEEE/ACM International Symposium on Microarchitecture*, pp. 763–783, 2022.

[4] H. Lee, M. Kim, D. Min, J. Kim, J. Back, H. Yoo, J.-H. Lee, and J. Kim, "3d-fpim: An extreme energy-efficient dnn acceleration system using 3d nand flash-based in-situ pim unit," in *2022 55th IEEE/ACM International Symposium on Microarchitecture*, pp. 1359–1376, 2022.

[5] J. Kim, M. Kang, Y. Han, Y.-G. Kim, and L.-S. Kim, "Optimstore:

In-storage optimization of large scale dnns with on-die processing," in *2023 IEEE International Symposium on High-Performance Computer Architecture*, pp. 611–623, 2023.

[6] C. Gao, X. Xin, Y. Lu, Y. Zhang, J. Yang, and J. Shu, "Parabit: Processing parallel bitwise operations in nand flash memory based ssds," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 59–70, 2021.

[7] N. Mansouri Ghiasi, J. Park, H. Mustafa, J. Kim, A. Olgun, A. Gollwitzer, D. Senol Cali, C. Firtina, H. Mao, N. Almadhoun Alserr, R. Ausavarungnirun, N. Vijaykumar, M. Alser, and O. Mutlu, "Genstore: a high-performance in-storage processing system for genome sequence analysis," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, p. 635–654, 2022.

[8] W. Xu, J. Kang, and T. Rosing, "A near-storage framework for boosted data preprocessing of mass spectrum clustering," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, p. 313–318, 2022.

[9] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, "Query processing on smart ssds: opportunities and challenges," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, p. 1221–1230, 2013.

[10] G. Koo, K. K. Matam, T. I., H. K. G. Narra, J. Li, H.-W. Tseng, S. Swanson, and M. Annavaram, "Summarizer: Trading communication with computing near storage," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 219–231, 2017.

[11] V. S. Mailthody, Z. Qureshi, W. Liang, Z. Feng, S. G. de Gonzalo, Y. Li, H. Franke, J. Xiong, J. Huang, and W.-m. Hwu, "Deepstore: In-storage acceleration for intelligent queries," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, p. 224–238, 2019.

[12] M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Catalina: In-storage processing acceleration for scalable big data analytics," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pp. 430–437, 2019.

[13] J. Jang, H.-S. Kim, W. Cho, H. Cho, J. Kim, S. I. Shim, Younggoan, J.-H. Jeong, B.-K. Son, D. W. Kim, Kihyun, J.-J. Shim, J. S. Lim, K.-H. Kim, S. Y. Yi, J.-Y. Lim, D. Chung, H.-C. Moon, S. Hwang, J.-W. Lee, Y.-H. Son, U.-I. Chung, and W.-S. Lee, "Vertical cell array using tcat (terabit cell array transistor) technology for ultra high density nand flash emory," in *Proceedings of the IEEE Symposium on VLSI Technology*, 2009.

[14] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories*. Springer, 2010.

[15] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 24–33, 2009.

[16] J. Park, M. Kim, M. Chun, L. Orosa, J. Kim, and O. Mutlu, "Reducing solid-state drive read latency by optimizing read-retry," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, p. 702–716, 2021.

[17] J. Maserjian and N. Zamani, "Behavior of the Si/SiO2 interface observed by Fowler-Nordheim tunneling," *Journal of Applied Physics*, vol. 53, no. 1, pp. 559–567, 1982.

[18] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-cosmos: In-flash bulk bitwise operations using inherent computation capability of nand flash memory," in *2022 55th IEEE/ACM International Symposium on Microarchitecture*, 2022.

[19] W. Cho, J. Jung, J. Kim, J. Ham, S. Lee, Y. Noh, D. Kim, W. Lee, K. Cho, K. Kim, H. Lee, S. Chai, E. Jo, H. Cho, J.-S. Kim, C. Kwon, C. Park, H. Nam, H. Won, T. Kim, K. Park, S. Oh, J. Ban, J. Park, J. Shin, T. Shin, J. Jang, J. Mun, J. Choi, H. Choi, S.-W. Choi, W. Park, D. Yoon, M. Kim, J. Lim, C. An, H. Shirr, H. Oh, H. Park,

S. Shim, H. Huh, H. Choi, S. Lee, J. Sim, K. Gwon, J. Kim, W. Jeong, J. Choi, and K.-W. Jin, "A 1-tb, 4b/cell, 176-stacked-wl 3d-nand flash memory with improved read latency and a 14.8gb/mm2 density," in *2022 IEEE International Solid-State Circuits Conference*, vol. 65, pp. 134–135, 2022.

[20] M. Kim, Y. Song, M. Jung, and J. Kim, "Saro: a state-aware reliability optimization technique for high density nand flash memory," in *Great Lakes Symposium on VLSI*, 2018.

[21] S. Han, M. Cho, G. Lee, and E.-Y. Chung, "Page type-aware data migration technique for read disturb management of nand flash memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 591–595, 2023.

[22] G. Zhang, Y. Deng, Y. Zhou, S. Pang, J. Yue, and Y. Zhu, "Cocktail: Mixing data with different characteristics to reduce read reclaims for nand flash memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2336–2349, 2022.

[23] J. Liao, J. Li, M. Zhao, Z. Sha, and Z. Cai, "Read refresh scheduling and data reallocation against read disturb in ssds," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 2, pp. 1–27, 2022.

[24] K. Ha, J. Jeong, and J. Kim, "An integrated approach for managing read disturbs in high-density nand flash memory," in *IEEE Transac-*

tions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, pp. 1079–1091, 2016.

[25] M. Zhao, J. Li, Z. Cai, J. Liao, and Y. Shi, "Block attribute-aware data reallocation to alleviate read disturb in ssds," in *2021 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1096–1099, 2021.

[26] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, "Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[27] C. Zou and A. A. Chien, "Assasin: Architecture support for stream computing to accelerate computational storage," in *2022 55th IEEE/ACM International Symposium on Microarchitecture*, pp. 354–368, 2022.

[28] S. Gupta, J. Morris, M. Imani, R. Ramkumar, J. Yu, A. Tiwari, B. Aksanli, and T. v. Rosing, "Thrifty: training with hyperdimensional computing across flash hierarchy," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020.

[29] S. Kim, Y. Jin, G. Sohn, J. Bae, T. J. Ham, and J. W. Lee, "Behemoth: A flash-centric training accelerator for extreme-scale DNNs," in *19th USENIX Conference on File and Storage Technologies*, pp. 371–385, 2021.

[30] M. Wilkening, U. Gupta, S. Hsia, C. Trippel, C.-J. Wu, D. Brooks, and G.-Y. Wei, "Recssd: near data processing for solid state drive based recommendation inference," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, p. 717–729, 2021.

[31] S.-W. Jun, A. Wright, S. Zhang, S. Xu, and Arvind, "Grafboost: Using accelerated flash storage for external graph analytics," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture*, pp. 411–424, 2018.

[32] K. K. Matam, G. Koo, H. Zha, H.-W. Tseng, and M. Annavaram, "Graphssd: graph semantics aware ssd," in *Proceedings of the 46th International Symposium on Computer Architecture*, p. 116–128, 2019.

[33] Y. Lee, J. Chung, and M. Rhu, "Smartsage: training large-scale graph neural networks using in-storage processing architectures," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, p. 932–945, 2022.

[34] D. Gouk, M. Kwon, H. Bae, and M. Jung, "Dockerssd: Containerized in-storage processing and hardware acceleration for computational ssds," in *2024 IEEE International Symposium on High-Performance Computer Architecture*, pp. 379–394, 2024.

[35] W. S. Jeong, C. Lee, K. Kim, M. K. Yoon, W. Jeon, M. Jung, and W. W. Ro, "React: Scalable and high-performance regular expression

pattern matching accelerator for in-storage processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1137–1151, 2020.

[36] S.-W. Jun, H. T. Nguyen, V. Gadepally, and Arvind, "In-storage embedded accelerator for sparse pattern processing," in *2016 IEEE High Performance Extreme Computing Conference*, pp. 1–7, 2016.

[37] Samsung, "Samsung 990 PRO." `https://www.techpowerup.com/ssd-specs/samsung-990-pro-1-tb.d861`.

[38] Micron, "Crucial T500." `https://www.techpowerup.com/ssd-specs/crucial-t500-1-tb.d1771`.

[39] S. Hynix, "SK Hynix Platinum P41." `https://www.techpowerup.com/ssd-specs/sk-hynix-platinum-p41-1-tb.d588`.

[40] T. Kouchi, M. Kakoi, N. Kumazaki, A. Sugahara, A. Imamoto, Y. Kajiyama, Y. Terada, B. Sanad, N. Kanagawa, T. Kodama, R. Fukuda, H. Komai, N. Asaoka, H. Ohnishi, R. Isomura, T. Handa, K. Yamamoto, Y. Ishizaki, Y. Deguchi, A. Okuyama, J. Sato, H. Yabe, H.-L. C. Hsu, and M. Yoshihara, "A 128gb 1-bit/cell 96-word-line-layer 3d flash memory to improve the random read latency with tprog = 75 µs and tr = 4 µs," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, pp. 225–234, 2021.

[41] B. Kim, S. Lee, B. Hah, K. Park, Y. Park, K. Jo, Y. Noh, H. Seol, H. Lee, J. Shin, S. Choi, Y. Jung, S. Ahn, Y. Park, S. Oh, M. Kim,

S. Kim, H. Park, T. Lee, H. Won, M. Kim, C. Koo, Y. Choi, S. Choi, S. Park, D. Youn, J. Lim, W. Park, H. Hur, K. Kwean, H. Choi, W. Jeong, S. Chung, J. Choi, and S. Cha, "28.2 a high-performance 1tb 3b/cell 3d-nand flash with a 194mb/s write throughput on over 300 layers i," in *2023 IEEE International Solid-State Circuits Conference*, pp. 27–29, 2023.

[42] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. D. Mundo, M. Rastegari, and M. Farajtabar, "Llm in a flash: Efficient large language model inference with limited memory," 2024.

[43] L. Guo, W. Choe, and F. X. Lin, "Sti: Turbocharge nlp inference at the edge via elastic pipelining," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, p. 791–803, 2023.

[44] P. Patel, E. Choukse, C. Zhang, A. Shah, Íñigo Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," 2024.

[45] J. Axboe, "Fio: Flexible i/o tester," 2006. `https://github.com/axboe/fio`.

[46] Samsung, "Communicating with your ssd understanding smart attributes," 2015. `https://web.archive.org/web/20150310051031/http://www.samsung.com/global/business/semiconductor/minisite/SSD/global/html/whitepaper/whitepaper07.html`.

[47] Micron, "Comparing SSD and HDD Endurance in the Age of QLC SSDs," 2018. https://www.micron.com/content/dam/micron/global/public/products/white-paper/5210-ssd-vs-hdd-endurance-white-paper.pdf.

[48] Samsung, "QLC SSD: High Density, High Performance and Affordable Solution for Client PCs," 2019. https://download.semiconductor.samsung.com/resources/white-paper/Samsung_SSD_860_QVO_White_paper.pdf.

[49] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?," 2019.

[50] S. Pang, Y. Deng, G. Zhang, J. Huang, and Z. Wu, "Minato: A read-disturb-aware dynamic buffer management scheme for nand flash memory," *IEEE TCAD*, 2024.

[51] D. Hong, M. Kim, G. Cho, D. Lee, and J. Kim, "GuardedErase: Extending SSD lifetimes by protecting weak wordlines," in *FAST*, 2022.

[52] Y. Shim, M. Kim, M. Chun, J. Park, Y. Kim, and J. Kim, "Exploiting process similarity of 3d flash memory for high performance ssds," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 211–223, 2019.

[53] J. Hruska, "How do ssds work? the inner details, explained," 2021. https://www.extremetech.com/gaming/210492-extremetech-explains-how-do-ssds-work.

[54] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proceedings of the IEEE*, 2017.

[55] ONFI Workgroup, "Open NAND Flash Interface Specification 5.1," 2022. `https://onfi.org/specs.html`.

[56] P. Chen, C. Zhang, H. Jiang, Z. Wang, and S. Yue, "High performance low complexity bch error correction circuit for ssd controllers," in *EDSSC*, 2015.

[57] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data retention in mlc nand flash memory: Characterization, optimization, and recovery," in *HPCA*, 2015.

[58] JEDEC, "JESD218B.02: Solid-State Drive (SSD) Requirements and Endurance Test Method," 2022. `https://www.jedec.org/standards-documents/docs/jesd218b01`.

[59] J.-W. Park, D. Kim, S. Ok, J. Park, T. Kwon, H. Lee, S. Lim, S.-Y. Jung, H. Choi, T. Kang, G. Park, C.-W. Yang, J.-G. Choi, G. Ko, J. Shin, I. Yang, J. Nam, H. Sohn, S.-I. Hong, Y. Jeong, S.-W. Choi, C. Choi, H.-S. Shin, J. Lim, D. Youn, S. Nam, J. Lee, M. Ahn, H. Lee, S. Lee, J. Park, K. Gwon, W. Jeong, J. Choi, J. Kim, and K.-W. Jin, "30.1 a 176-stacked 512gb 3b/cell 3d-nand flash with 10.8gb/mm2 density with a peripheral circuit under cell array architecture," in *2021 IEEE International Solid-State Circuits Conference*, vol. 64, pp. 422–423, 2021.

[60] M. Lapedus, "3D NAND flash wars begin," 2018. https://semiengineering.com/3d-nand-flash-wars-begin/.

[61] A. S. Spinelli, G. Malavena, A. L. Lacaita, and C. Monzio Compagnoni, "Random telegraph noise in 3d nand flash memories," *Micromachines*, 2021.

[62] W.-C. Chen, H.-T. Lue, K.-P. Chang, Y.-H. Hsiao, C.-C. Hsieh, Y.-H. Shih, and C.-Y. Lu, "Study of the programming sequence induced back-pattern effect in split-page 3d vertical-gate (vg) nand flash," in *VLSI-TSA*, 2014.

[63] Q. Li, M. Ye, Y. Cui, L. Shi, X. Li, T.-W. Kuo, and C. J. Xue, "Shaving retries with sentinels for fast read over high-density 3d flash," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 483–495, 2020.

[64] C. Kim, D.-H. Kim, W. Jeong, H.-J. Kim, I. H. Park, H.-W. Park, J. Lee, J. Park, Y.-L. Ahn, J. Y. Lee, S.-B. Kim, H. Yoon, J. D. Yu, N. Choi, N. Kim, H. Jang, J. Park, S. Song, Y. Park, J. Bang, S. Hong, Y. Choi, M.-S. Kim, H. Kim, P. Kwak, J.-D. Ihm, D. S. Byeon, J.-Y. Lee, K.-T. Park, and K.-H. Kyung, "A 512-Gb 3-b/Cell 64-Stacked WL 3-D-NAND Flash Memory," *IEEE JSSC*, vol. 53, no. 1, pp. 124–133, 2018.

[65] S.-H. Kim, J. Shim, E. Lee, S. Jeong, I. Kang, and J.-S. Kim, "NVMeVirt: A versatile software-defined virtual NVMe device," in

21st USENIX Conference on File and Storage Technologies, (Santa Clara, CA), pp. 379–394, 2023.

[66] J. Cho, D. C. Kang, J. Park, S.-W. Nam, J.-H. Song, B.-K. Jung, J. Lyu, H. Lee, W.-T. Kim, H. Jeon, S. Kim, I.-M. Kim, J.-I. Son, K. Kang, S.-W. Shim, J. Park, E. Lee, K.-M. Kang, S.-W. Park, J. Lee, S. H. Moon, P. Kwak, B. Jeong, C. A. Lee, K. Kim, J. Ko, T.-H. Kwon, J. Lee, Y. Lee, C. Kim, M.-W. Lee, J.-y. Yun, H. Lee, Y. Choi, S. Hong, J. Park, Y. Shin, H. Kim, H. Kim, C. Yoon, D. S. Byeon, S. Lee, J.-Y. Lee, and J. Song, "30.3 A 512Gb 3b/Cell 7 th-Generation 3D-NAND Flash Memory with 184MB/s Write Throughput and 2.0 Gb/s Interface," in *Proceedings of the 2021 IEEE International Solid-State Circuits Conference*, 2021.

[67] Samsung, "Samsung enterprise SSDs," 2023. `https://semiconductor.samsung.com/ssd/enterprise-ssd`.

[68] SK Hynix, "SK Hynix enterprise SSDs," 2023. `https://product.skhynix.com/products/ssd/essd.go`.

[69] R. McDougall and J. Mauro, "Filebench," *URL: http://www.nfsv4bat. org/Documents/nasconf/2004/filebench. pdf (Cited on page 56.)*, 2005.

[70] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, (New York, NY, USA), p. 143–154, 2010.

[71] C.-Y. Liu, Y. Lee, M. Jung, M. T. Kandemir, and W. Choi, "Prolonging 3d nand ssd lifetime via read latency relaxation," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[72] S. Wang, Z. Lin, S. Wu, H. Jiang, J. Zhang, and B. Mao, "Learnedftl: A learning-based page-level ftl for reducing double reads in flash-based ssds," in *2024 IEEE International Symposium on High-Performance Computer Architecture*, 2024.

[73] Z. Jiao, X. Zhang, H. Shin, J. Choi, and B. S. Kim, "The design and implementation of a Capacity-Variant storage system," in *22nd USENIX Conference on File and Storage Technologies*, 2024.

[74] Y. Wang, Y. Zhou, F. Wu, Y. Zhong, J. Zhou, Z. Lu, S. Li, Z. Wang, and C. Xie, "Holistic and opportunistic scheduling of background i/os in flash-based ssds," *IEEE Transactions on Computers*, 2023.

[75] C. Duffy, J. Shim, S.-H. Kim, and J.-S. Kim, "Dotori: A key-value ssd based kv store," 2023.

[76] Dharamjeet, Y.-S. Chen, T.-Y. Chen, Y.-H. Kuan, and Y.-H. Chang, "LLSM: A Lifetime-Aware Wear-Leveling for LSM-Tree on NAND Flash Memory," *IEEE TCAD*, 2022.

[77] H. Li, M. L. Putra, R. Shi, F. I. Kurnia, X. Lin, J. Do, A. I. Kistijantoro, G. R. Ganger, and H. S. Gunawi, "Extending and Programming the NVMe I/O Determinism Interface for Flash Arrays," *ACM TOS*, 2023.

[78] Y. Jun, S. Park, J.-U. Kang, S.-H. Kim, and E. Seo, "We Ain't Afraid of No File Fragmentation: Causes and Prevention of Its Performance Impact on Modern Flash SSDs," in *FAST*, 2024.

[79] S. Oh, J. Kim, S. Han, J. Kim, S. Lee, and S. H. Noh, "MIDAS: Minimizing Write Amplification in Log-Structured Systems through Adaptive Group Number and Size Configuration," in *FAST*, 2024.

[80] Y. Lv, L. Shi, Q. Li, C. Gao, Y. Song, L. Luo, and Y. Zhang, "MGC: Multiple-Gray-Code for 3D NAND Flash based High-Density SSDs," in *HPCA*, 2023.

[81] J. Lee, M. Kim, W. Choi, S. Lee, and J. Kim, "Tailcut: improving performance and lifetime of ssds using pattern-aware state encoding," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, p. 409–414, 2022.

[82] Y. Fu, L. Xue, Y. Huang, A.-O. Brabete, D. Ustiugov, Y. Patel, and L. Mai, "{ServerlessLLM}:{Low-Latency} serverless inference for large language models," in *18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 135–153, 2024.

[83] H. Jang, J. Song, J. Jung, J. Park, Y. Kim, and J. Lee, "Smart-infinity: Fast large language model training using near-storage processing on a real system," in *2024 IEEE International Symposium on High-Performance Computer Architecture*, pp. 345–360, IEEE, 2024.

[84] H. Liu and H. H. Huang, "Graphene: Fine-Grained IO Management for Graph Computing," in *FAST*, 2017.

[85] N. Elyasi, C. Choi, and A. Sivasubramaniam, "Large-Scale Graph Processing on Emerging Storage Devices," in *FAST*, 2019.

[86] S.-H. Park, D. Kwon, H.-N. Yoo, J.-W. Back, J. Hwang, Y. Yang, J.-J. Kim, and J.-H. Lee, "Retention improvement in vertical nand flash memory using 1-bit soft erase scheme and its effects on neural networks," in *2022 International Electron Devices Meeting*, pp. 5.5.1–5.5.4, 2022.

[87] S. Mishra, H. Amrouch, J. Joe, C. K. Dabhi, K. Thakor, Y. S. Chauhan, J. Henkel, and S. Mahapatra, "A simulation study of nbti impact on 14-nm node finfet technology for logic applications: Device degradation to circuit-level interaction," *IEEE Transactions on Electron Devices*, vol. 66, no. 1, pp. 271–278, 2019.

[88] C.-T. Tung and C. Hu, "Neural network-based bsim transistor model framework: Currents, charges, variability, and circuit simulation," *IEEE Transactions on Electron Devices*, vol. 70, no. 4, pp. 2157–2160, 2023.

[89] *BSIM-CMG Technical Manual*, 2020. Accessed: Apr. 1, 2020. [Online]. Available: `http://bsim.berkeley.edu/models/bsimcmg/`.

[90] J. P. Duarte, S. Khandelwal, A. Medury, C. Hu, P. Kushwaha, H. Agarwal, A. Dasgupta, and Y. S. Chauhan, "Bsim-cmg: Standard finfet compact model for advanced circuit design," in *ESSCIRC*

Conference 2015 - 41st European Solid-State Circuits Conference, pp. 196–201, 2015.

[91] Cadence, "Virtuoso." https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-design/virtuoso-schematic-editor.html.

[92] H. Makino and T. Tanzawa, "A 30

[93] J. Kondo and T. Tanzawa, "Pre-emphasis pulse design for reducing bit-line access time in nand flash memory," *Electronics*, vol. 11, 2022.

[94] M.-Y. Kao, F. Chavez, S. Khandelwal, and C. Hu, "Deep learning-based bsim-cmg parameter extraction for 10-nm finfet," *IEEE Transactions on Electron Devices*, vol. 69, no. 8, pp. 4765–4768, 2022.

[95] Cadence, "Spectre." https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-design/virtuoso-schematic-editor.html.

[96] H. S. Jeong and S. H. Cho, "Word-line and charge-pump modeling of nand flash using standard cmos logic process," *Journal of Integrated Circuits and Systems*, vol. 7, no. 4, 2021.

[97] O. Khouri, R. Micheloni, A. Sacco, G. Campardo, and G. Torelli, "Program word-line voltage generator for multilevel flash memories," in *ICECS 2000. 7th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.00EX445)*, vol. 2, pp. 1030–1033 vol.2, 2000.

[98] S. Won, Y. Noh, H. Cho, J. Ryu, S.-U. Choi, S. Choi, D. Kim, J. Chung, B.-S. Han, and E.-Y. Chung, "High-voltage wordline generator for low-power program operation in nand flash memories," *IEEE Asian Solid-State Circuits Conference 2011*, pp. 169–172, 2011.

[99] S. Cho, B. Kim, H. Cho, G. Seo, O. Mutlu, M. Kim, and J. Park, "Aero: Adaptive erase operation for improving lifetime and performance of modern nand flash-based ssds," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, p. 101–118, Association for Computing Machinery, 2024.

[100] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang, "LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives," in *11th USENIX Conference on File and Storage Technologies*, pp. 243–256, 2013.

[101] S.-H. Kuo, "Ultra MMI : an LDPC decoder that doubles throughput at end-of-life.," in *Flash Memory Summit*, 2019.

[102] L.-W. Liu, M.-H. Yuan, Y.-C. Liao, and H.-C. Chang, "A 38.64-gb/s large-cpm 2-kb ldpc decoder implementation for nand flash memories," *IEEE Open Journal of Circuits and Systems*, vol. 3, pp. 180–191, 2022.

[103] M. Kim, Y. Song, M. Jung, and J. Kim, "Saro: A state-aware reliability optimization technique for high density nand flash memory," in

Proceedings of the 2018 Great Lakes Symposium on VLSI, GLSVLSI '18, p. 255–260, 2018.

[104] M. Ye, Q. Li, C. Gao, S. Deng, T.-W. Kuo, and C. J. Xue, "Stop unnecessary refreshing: extending 3d nand flash lifetime with orber," *CCF Trans. High Perform. Comput.*, vol. 4, no. 3, pp. 281–301, 2022.

[105] Y. Lv, L. Shi, Q. Li, C. Gao, Y. Song, L. Luo, and Y. Zhang, "Mgc: Multiple-gray-code for 3d nand flash based high-density ssds," in *2023 IEEE International Symposium on High-Performance Computer Architecture*, pp. 122–136, 2023.

[106] J.-W. Park, D. Kim, S. Ok, J. Park, T. Kwon, H. Lee, S. Lim, S.-Y. Jung, H. Choi, T. Kang, G. Park, C.-W. Yang, J.-G. Choi, G. Ko, J. Shin, I. Yang, J. Nam, H. Sohn, S.-I. Hong, Y. Jeong, S.-W. Choi, C. Choi, H.-S. Shin, J. Lim, D. Youn, S. Nam, J. Lee, M. Ahn, H. Lee, S. Lee, J. Park, K. Gwon, W. Jeong, J. Choi, J. Kim, and K.-W. Jin, "30.1 a 176-stacked 512gb 3b/cell 3d-nand flash with 10.8gb/mm2 density with a peripheral circuit under cell array architecture," in *2021 IEEE International Solid-State Circuits Conference*, vol. 64, pp. 422–423, 2021.

[107] J.-W. Im, W.-P. Jeong, D.-H. Kim, S.-W. Nam, D.-K. Shim, M.-H. Choi, H.-J. Yoon, D.-H. Kim, Y.-S. Kim, H.-W. Park, D.-H. Kwak, S.-W. Park, S.-M. Yoon, W.-G. Hahn, J.-H. Ryu, S.-W. Shim, K.-T. Kang, S.-H. Choi, J.-D. Ihm, Y.-S. Min, I.-M. Kim, D.-S. Lee, J.-H. Cho, O.-S. Kwon, J.-S. Lee, M.-S. Kim, S.-H. Joo, J.-H. Jang, S.-W.

Hwang, D.-S. Byeon, H.-J. Yang, K.-T. Park, K.-H. Kyung, and J.-H. Choi, "7.2 a 128gb 3b/cell v-nand flash memory with 1gb/s i/o rate," in *2015 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pp. 1–3, 2015.

[108] R. Yi, L. Guo, S. Wei, A. Zhou, S. Wang, and M. Xu, "Edgemoe: Fast on-device inference of moe-based large language models," 2023.

[109] D. Xu, W. Yin, X. Jin, Y. Zhang, S. Wei, M. Xu, and X. Liu, "Llmcad: Fast and scalable on-device large language model inference," 2023.

[110] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020.

[111] L. Yan, H. Cynthia, and O. Ken, "Non-Volatile Memory And Method With Improved First Pass Programming." U.S Patent 8811091, 2014.

[112] J. Park, J. Jeong, S. Lee, Y. Song, and J. Kim, "Improving performance and lifetime of nand storage systems using relaxed program sequence," in *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, Association for Computing Machinery, 2016.

[113] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul, "Spdk: A development kit to build high performance storage applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science*, pp. 154–161, 2017.

[114] N. Express, "NVMe Namespace." https://nvmexpress.org/resource/nvme-namespaces/.

[115] S.-H. Kim, J. Shim, E. Lee, S. Jeong, I. Kang, and J.-S. Kim, "NVMeVirt: A versatile software-defined virtual NVMe device," in *21st USENIX Conference on File and Storage Technologies*, (Santa Clara, CA), pp. 379–394, 2023.

[116] G. Gerganov, "ggerganov/llama. cpp: Port of facebook's llama model in c/c++," 2023.

[117] M. Llama, "LLaMA2-7B." https://huggingface.co/meta-llama/Llama-2-7b-hf.

[118] M. Llama, "LLaMA3-8B." https://huggingface.co/meta-llama/Meta-Llama-3-8B.

[119] T. I. Institute, "Falcon-11B." https://huggingface.co/tiiuae/falcon-11B.

[120] M. Llama, "LLaMA2-13B." https://huggingface.co/meta-llama/Llama-2-13b-hf.

[121] M. AI, "Mixtral-8x7B." https://huggingface.co/mistralai/Mixtral-8x7B-v0.1.

[122] EleutherAI, "GPT-NeoX-20B." https://huggingface.co/EleutherAI/gpt-neox-20b.

[123] T. I. Institute, "Falcon-40B." https://huggingface.co/tiiuae/falcon-40b.

[124] M. Llama, "LLaMA3-70B." https://huggingface.co/meta-llama/Meta-Llama-3-70B.

[125] S. M. Lee, H. Kim, J. Yeon, J. Lee, Y. Choi, M. Kim, C. Park, K. Jang, Y. Kim, Y. Kim, C. Lee, H. Han, W. E. Kim, R. Tang, and J. H. Baek, "A 64-tops energy-efficient tensor accelerator in 14nm with reconfigurable fetch network and processing fusion for maximal data reuse," *IEEE Open Journal of the Solid-State Circuits Society*, vol. 2, pp. 219–230, 2022.

[126] T. P. J., "Prospects for memory," in *Workshop on Memory Centric High Performance Computing*, 2019.

[127] D. Na, J.-w. Lee, S.-K. Lee, H. Cho, J. Lee, M. Yang, E. Song, A. Kavala, T. Kim, D.-S. Jang, Y. Jo, J.-Y. Shin, B.-K. Chun, T.-s. Lee, B. Jeong, C.-W. Yoon, D. Kang, S. Lee, J. Ihm, D. S. Byeon, J. Lee, and J. H. Song, "A 1.8-gb/s/pin 16-tb nand flash memory multi-chip package with f-chip for high-performance and high-capacity storage," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1129–1140, 2021.

[128] S. Li, F. Spagna, J. Chen, X. Wang, L. Tong, S. Gowder, W. Jia, R. Nicholson, S. Iyer, R. Song, L. Li, M.-h. Chen, A. Tran, M. D. Vita, D. Govindrajan, M. Pasquarella, D. Bradley, F. Verdico, M. Duwe, E. Lee, and M. Wigton, "A power and area efficient 2.5-16 gbps gen4 pcie phy in 10nm finfet cmos," in *2018 IEEE Asian Solid-State Circuits Conference*, pp. 5–8, 2018.