Pusan National University Computer
Science and Engineering Technical Report
2025-09

딥러닝을 이용한 SSD 성능 개선 연구

201924544 이준형

201914502 강인석

202155537 김지수

지도교수 안성용

목 차

1. 서론

1.1. 연구 배경

현대 컴퓨팅 환경에서 SSD(Solid-State Drive)는 빠른 입출력 속도와 낮은 지연 시간 덕분에 주요 저장 장치로 자리 잡았다. 하지만 SSD의 핵심 저장 매체인 낸드 플래시 메모리(NAND flash memory)는 구조적인 한계를 지닌다. 데이터를 덮어쓸 수 없으며, 읽기와 쓰기는 페이지(page) 단위로 이루어지지만, 삭제는 이보다 훨씬 큰 블록(block) 단위로만 가능하다는 점이다. 이로 인해 SSD는 기존의 데이터를 무효화하고 새로운 데이터를 다른 공간에 쓰는 방식으로 쓰기 작업을 수행한다. 이 과정이 반복되면 유효하지 않은 데이터가 누적되고, 결국 가비지 컬렉션(Garbage Collection, GC)이라는 정리 작업을 통해 유효한 데이터를 다른 블록으로 옮기는 추가적인 쓰기가 발생한다. 이러한 문제점을 해결하기 위해 SSD는 FTL(Flash Translation Layer)이라는 펌웨어 계층을 통해 논리 주소-물리 주소 매핑, GC, 웨어 레벨링(Wear Leveling) 등을 관리한다.

1.2. 기존 문제점

가비지 컬렉션 과정에서 발생하는 추가적인 쓰기 작업은 쓰기 증폭(Write Amplification Factor, WAF)이라는 현상을 유발한다. 이는 실제 사용자 요청보다 더 많은 양의 데이터가 SSD 내부에 기록되어 SSD의 성능을 저하하고, 각 블록의 쓰기 횟수를 제한하는 P/E(Program/Erase) 사이클을 빠르게 소모하여 SSD의 수명을 단축시킨다. 하지만 대부분의 FTL은 사전에 정의된 정적인 정책을 따르기 때문에, 사용자의 다양한 입출력 패턴에 유연하게 대응하지 못해 쓰기 증폭 문제를 효과적으로 해결하지 못하고 있다.

1.3. 연구 목표

본 연구는 위 문제점들을 해결하기 위해 딥러닝 기술을 도입하여 SSD 성능을 개선하는 방안을 제안한다. 입출력 워크로드의 패턴을 분석하고, 딥러닝 모델을 활용해 데이터의 Hotness(갱신 빈도)를 예측한다. 이렇게 예측된 Hotness 정보를 기반으로 FTL이 데이터를 블록별로 분리 저장하는 동적 정책을 구현함으로써, 가비지 컬렉션 시 유효 페이지를 옮기는 횟수를 최소화하여 쓰기 증폭을 완화하는 것을 목표로 한다. 최종적으로 시뮬레이터를 이용한 정량적 실험을 통해 기존 FTL 대비 쓰기 증폭 감소 및 SSD 수명 향상 효과를 검증하는 것을 목표로 한다.

2. 연구 배경

2.1. SSD의 구조 및 데이터 관리 방식

현대 컴퓨팅 시스템의 핵심 저장 장치로 자리 잡은 SSD(Solid-State Drive)는 낸드 플래시 메모리(NAND Flash Memory)를 기반으로 동작한다. 낸드 플래시는 D램과 달리 비휘발성 특성을 가지면서도 HDD(Hard Disk Drive)보다 월등히 빠른 입출력 속도를 제공하지만, 동시에 구조적인 제약을 내포한다. 가장 핵심적인 제약은 덮어쓰기 불가(No Overwrite) 원칙과 읽기/쓰기 단위와 삭제 단위의 불일치이다. 데이터는 작은 단위인 페이지(Page)를 기준으로 읽고 쓰지만, 삭제는 수백 개의 페이지가 모인 블록(Block) 단위로만 수행할 수 있다. 이러한 특성으로 인해 SSD는 데이터를 수정할 때, 기존 데이터를 직접 덮어쓰지 않고 새로운 위치의 빈 페이지에 변경된 데이터를 쓰는 Out-of-Place-Write 방식을 사용한다. 이 과정에서 기존 데이터가 저장된 페이지는 더 이상 유효하지 않은 무효(Invalid) 상태로 표시된다. 쓰기 작업이 반복되면 SSD 내부에는 유효한 데이터와 무효한 데이터가 혼재하게 되고, 새로운 데이터를 기록할 빈 공간이 부족해진다. 이때 가비지 컬렉션(Garbage Collection, GC)이라는 내부 관리 프로세스가 필수적으로 요구된다. GC는 여러 블록에 흩어져 있는 유효한 페이지만을 새로운 빈 블록으로 복사(Copy)한 뒤, 원래 블록에 남아있던 모든 데이터(유효 및 무효 페이지)를 삭제(Erase)하여 다시 사용할 수 있는 깨끗한 블록으로 만드는 과정이다. 이 모든 복잡한 데이터 관리와 주소 변환은 FTL(Flash Translation Layer) 이라는 펌웨어 계층이 전담하여 처리한다. FTL은 호스트 시스템의 논리 주소(LBA)를 낸드 플래시의 물리 주소(PPA)로 매핑하고, GC와 웨어 레벨링(Wear-Leveling)을 수행하며 SSD의 효율적인 동작을 보장하는 핵심 요소이다.

2.2. 기존 FTL의 한계 및 문제점

FTL의 데이터 관리, 특히 GC 과정은 SSD 성능에 있어 양날의 검과 같다. GC는 SSD의 지속적인 사용을 가능하게 하지만, 이 과정에서 발생하는 추가적인 내부 쓰기는 쓰기 증폭(Write Amplification Factor, WAF)이라는 고질적인 문제를 유발한다. 쓰기 증폭은 사용자가 실제로 요청한 쓰기 데이터의 양보다 훨씬 많은 양의 데이터가 SSD 내부에 물리적으로 기록되는 현상을 의미하며, 이는 다음과 같은 심각한 부작용을 초래한다.

첫째, 성능 저하이다. GC 수행 시 발생하는 유효 페이지 복사 작업은 내부 I/O 대역폭을 소모하며, 이로 인해 사용자 요청에 대한 응답 시간이 길어지는 지연(Latency) 현상이

발생한다.

둘째, 수명 단축이다. 낸드 플래시의 각 셀은 프로그램/삭제(Program/Erase, P/E) 사이클 횟수가 제한되어 있다. 높은 쓰기 증폭은 불필요한 쓰기 작업을 가중시켜 P/E 사이클을 빠르게 소모시키고, 결과적으로 SSD의 전체 수명을 단축시키는 결정적인 원인이 된다.

이러한 문제의 근본 원인은 대부분의 상용 FTL이 사전에 정의된 정적(Static) 정책에 기반하여 동작하기 때문이다. 정적 FTL은 데이터의 특성이나 접근 패턴의 변화를 고려하지 않고 고정된 규칙에 따라 GC Victim 블록을 선정하고 데이터를 배치한다. 이로 인해 실제 사용 환경에서 발생하는 다양하고 동적인 입출력 워크로드에 유연하게 대응하지 못하여, 비효율적인 GC를 반복하고 높은 쓰기 증폭을 유발하는 한계를 가진다.

2.3. 딥러닝 기반 FTL 연구의 필요성

기존 FTL의 정적 정책이 야기하는 한계를 극복하기 위해서는 데이터의 특성을 파악하고 미래의 접근 패턴을 예측하여 FTL 정책을 동적으로 최적화하는 지능형 접근법이 요구된다. 데이터의 Hotness는 이러한 FTL 최적화의 핵심적인 지표이다. 자주 갱신되는 Hot 데이터와 거의 변경되지 않는 Cold 데이터를 구분하여 관리할 경우, GC의 효율을 극대화할 수 있기 때문이다. 예를 들어, Hot 데이터만을 별도의 블록에 모아두면 해당 블록은 무효 페이지가 빠르게 누적되어 적은 유효 페이지 복사만으로도 효율적인 GC가 가능해진다.

그러나 데이터의 Hotness는 워크로드에 따라 실시간으로 변하기 때문에, 정적 규칙만으로는 이를 정확히 식별하고 분리하기 어렵다. 바로 이 지점에서 딥러닝 기술의 필요성이 대두된다. 딥러닝 모델, 특히 시계열 데이터 분석에 강점을 보이는 LSTM(Long Short-Term Memory)과 같은 모델은 복잡한 I/O 트레이스의 순차적 패턴을 학습하여 각 데이터 페이지의 미래 Hotness를 높은 정확도로 예측할 수 있다.

따라서 본 연구는 딥러닝 모델을 통해 예측된 Hotness 정보를 FTL의 데이터 배치 및 GC 정책에 능동적으로 반영하는 새로운 FTL 아키텍처를 제안한다. 예측된 Hotness에 따라 데이터를 물리적으로 분리 저장함으로써, GC 시 발생하는 불필요한 유효 페이지 복사를 최소화하고 쓰기 증폭을 근본적으로 완화하는 것이 가능하다. 최종적으로 이는 쓰기 지연 시간 단축과 P/E 사이클 소모 절감을 통해

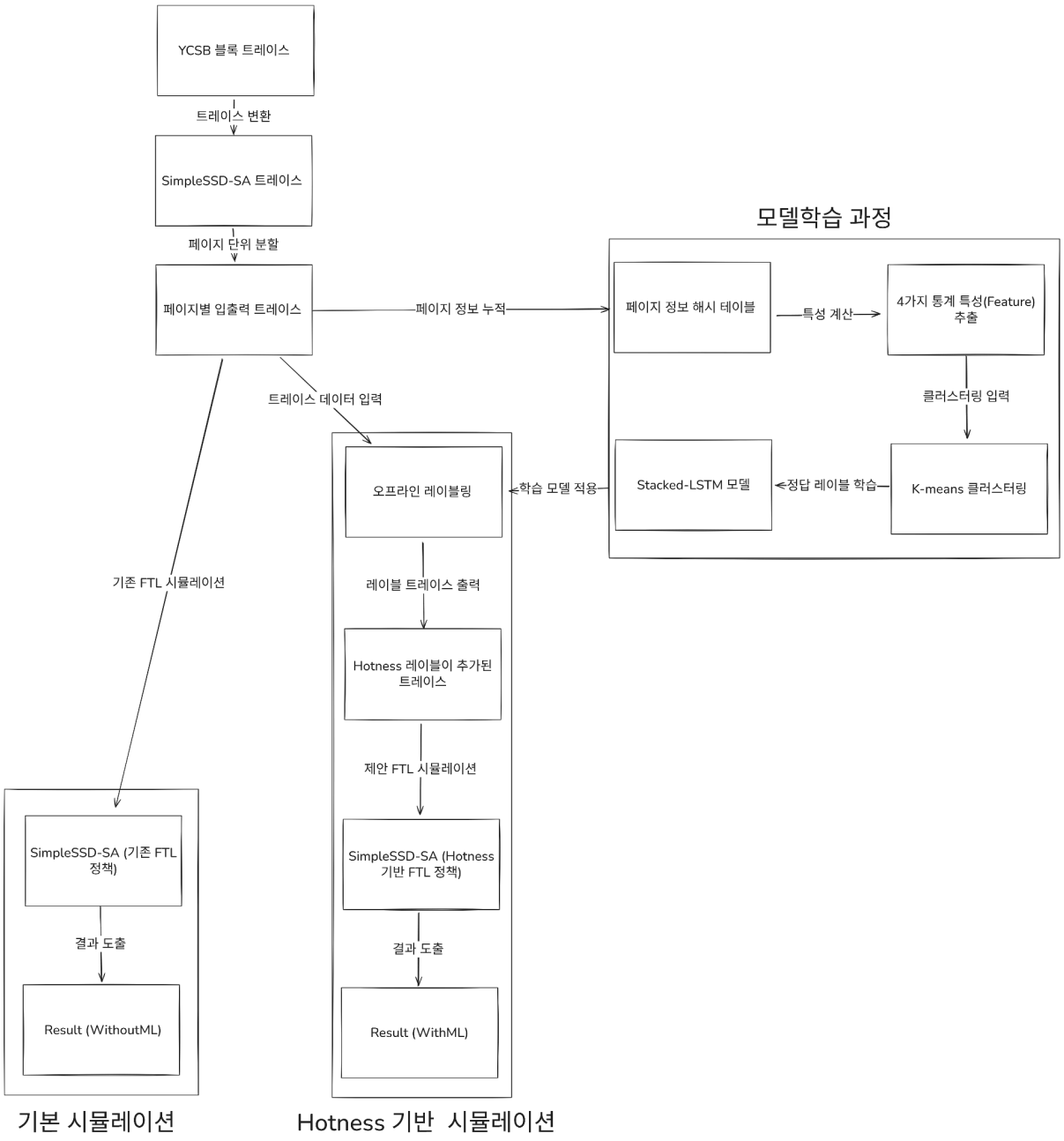
SSD의 성능과 수명을 동시에 향상시키는 효과로 이어질 것이며, 본 연구는 시뮬레이션을

통해 이러한 딥러닝 기반 FTL의 실효성과 잠재력을 정량적으로 검증하는 것을 목표로 한다.

3. 연구 내용

3.1. 전체 연구 개요

딥러닝을 활용하여 SSD의 Hotness 기반 FTL 정책을 제안하고 검증한다. 연구는 먼저 원본 YCSB 블록 트레이스를 SimpleSSD-SA[1] 시뮬레이터에 맞는 형식으로 정제하고, 이를 페이지별로 분할하여 페이지별 입출력 트레이스를 생성하는 데이터 전처리 과정으로 시작한다. 이 페이지별 트레이스를 기반으로 페이지 정보 해시 테이블을 구축하여 각 페이지의 접근 패턴에 대한 통계 정보를 누적한다. 이 해시 테이블에서 추출된 4가지 통계 특성(평균 시간 간격, 시간 간격 표준편차, 평균 요청 크기, 마지막 수명)은 K-means 클러스터링을 통해 Hotness 레이블을 생성하는 데 사용되며, 동시에 Stacked-LSTM 모델의 입력 데이터가 되어 페이지 접근 패턴의 시계열적 특성을 학습한다. 학습이 완료된 모델은 오프라인 레이블링을 통해 전체 트레이스에 Hotness 예측 레이블을 추가하여 시뮬레이션에 사용될 Hotness 이 추가된 트레이스를 생성한다. 최종적으로, SimpleSSD-SA 시뮬레이터를 사용하여 Hotness를 고려하지 않는 기존 FTL 정책과 제안된 Hotness 기반 FTL 정책의 성능을 비교 분석한다.



[그림 1] 실험 설계 전체 개요

3.2. 데이터 전처리 및 학습 데이터 생성

3.2.1. 입출력 트레이스 변환

시뮬레이터의 재현성과 시뮬레이션 효율성을 확보하기 위해, SNIA IOTTA 리포지터리에서 제공하는 YCSB 블록 트레이스[2]에 대한 전처리 과정이 필수적이다. 이 과정에서 원본 트레이스의 불필요한 식별자와 메타데이터를 제거하고, 시뮬레이터의 일관된 동작을 위해 통일된 형식으로 정제한다. 구체적인 변환 과정은 다음과 같다.

- **Device, Seq, PID**: 원본 트레이스의 다양한 값을 "8,0", 0, 0과 같은 더미 값으로 고정하여 외부 환경 정보에 대한 의존성을 제거한다.
- **Timestamp**: 초 단위 값의 소수점 이하를 9자리까지 패딩 처리함으로써 입출력 요청 간 시간 간격을 정밀하게 반영한다.
- **Op (Operation Type)**: R/RS(읽기)를 R로, W/WS(쓰기)를 W로 단순화하여 시뮬레이터의 입력 범위를 읽기와 쓰기 두 가지 유형으로 제한한다.
- **ProcessName**: 출력에서 완전히 제거하여 데이터 크기를 줄이고 파싱 로직의 복잡도를 낮춘다.

이러한 변환을 통해 입력 데이터의 크기를 줄이고 파싱 및 처리 로직을 간소화하며, Sector와 Size 등 핵심 정보를 보존함으로써 시뮬레이터가 원본 워크로드의 특성을 정확하게 반영하도록 보장한다.

[표 1] 입출력 데이터 필드 변환 정리

YCSB입력 필드	SimpleSSD-SA 출력 필드	변환 방식 및 설명
Device	Device	Major,Minor 번호를 “8,0”의 더미값으로 고정
CPU	CPU	값 변경 없이 그대로 사용
RecordID	Seq	값 제거, 출력을 “0”(더미값)으로 고정
Timestamp	Timestamp	초 단위 값에 대해 소수점 이하 9자리까지 패딩(예: 0.000005462 → 0.000005462)
PID	PID	값 제거, 출력을 “0”(더미값)으로 고정
Action	Action	D(issued)만 유지, 그대로 출력
Op	Op	R/RS → R, W/WS → W로 단순화
Sector	Sector	값 변경 없이 그대로 사용
+	+	값 변경 없이 그대로 사용
Size	Size	값 변경 없이 그대로 사용
ProcessName	- (없음)	출력에서 완전히 제거

[표 2] 입력 데이터 예시 (YCSB)

Device	CPU	RecordID	Timestamp	PID	Action	Op	Sector	+	Size	ProcessName
259,2	0	5	0.000005462	4020	D	RS	282624	+	8	[java]
259,2	0	11	0.013363110	4020	D	RS	286720	+	128	[java]

[표 3] 출력 데이터 예시 (SimpleSSD-SA)

Device	CPU	Seq	Timestamp	PID	Action	Op	Sector	+	Size
8,0	0	0	0.000005462	0	D	R	282624	+	8
8,0	0	0	0.013363110	0	D	R	286720	+	128

3.2.2. 페이지 단위 분할

딥러닝 모델이 페이지 단위의 접근 패턴을 학습하도록 하기 위해서는, 대용량 입출력 요청을 SSD의 물리적 페이지 경계에 맞춰 세분화하는 과정이 필수적이다. 이 과정을 통해 페이지별 해시 테이블 업데이트 및 Hotness 레이블링에 활용할 수 있는 형태로 데이터를 변환한다.

구체적인 분할 절차는 다음과 같다. 요청 크기가 기본 페이지 단위인 8섹터의 배수로 주어지면, 이를 8섹터씩 나누어 여러 개의 서브 요청으로 분해한다. 각 서브 요청의 시작 LBA(Logical Block Address)는 원본 Sector 값에 (페이지 인덱스 × 8)을 더해 계산하며, Size는 8로 고정한다. 이로써 하나의 대용량 요청은 동일한 Timestamp, Seq, Op를 유지한 채 여러 개의 페이지별 요청으로 분해되어, 이후 페이지 단위 처리 로직에 즉시 투입될 수 있도록 준비한다.

[표 4] 입력 데이터 예시 (SimpleSSD-SA 트레이스)

Device	CPU	Seq	Timestamp	PID	Action	Op	Sector	+	Size
8,0	0	0	150.122992736	0	D	R	381785824	+	256

[표 5] 출력 데이터 예시 (페이지별 분할 트레이스)

Device	CPU	Seq	Timestamp	PID	Action	Op	Sector	+	Size
8,0	0	0	150.122992736	0	D	R	381785824	+	8
8,0	0	0	150.122992736	0	D	R	381785832	+	8
8,0	0	0	150.122992736	0	D	R	381785840	+	8
8,0	0	0	150.122992736	0	D	R	381785848	+	8

8,0	0	0	150.122992736	0	D	R	381785856	+	8
8,0	0	0	150.122992736	0	D	R	381785864	+	8
8,0	0	0	150.122992736	0	D	R	381785872	+	8
8,0	0	0	150.122992736	0	D	R	381785880	+	8
8,0	0	0	150.122992736	0	D	R	381785888	+	8
8,0	0	0	150.122992736	0	D	R	381785896	+	8
8,0	0	0	150.122992736	0	D	R	381785904	+	8
8,0	0	0	150.122992736	0	D	R	381785912	+	8
8,0	0	0	150.122992736	0	D	R	381785920	+	8
8,0	0	0	150.122992736	0	D	R	381785928	+	8
8,0	0	0	150.122992736	0	D	R	381785936	+	8
8,0	0	0	150.122992736	0	D	R	381785944	+	8
8,0	0	0	150.122992736	0	D	R	381785952	+	8
8,0	0	0	150.122992736	0	D	R	381785960	+	8
8,0	0	0	150.122992736	0	D	R	381785968	+	8
8,0	0	0	150.122992736	0	D	R	381785976	+	8
8,0	0	0	150.122992736	0	D	R	381785984	+	8
8,0	0	0	150.122992736	0	D	R	381785992	+	8
8,0	0	0	150.122992736	0	D	R	381786000	+	8
8,0	0	0	150.122992736	0	D	R	381786008	+	8
8,0	0	0	150.122992736	0	D	R	381786016	+	8
8,0	0	0	150.122992736	0	D	R	381786024	+	8
8,0	0	0	150.122992736	0	D	R	381786032	+	8
8,0	0	0	150.122992736	0	D	R	381786040	+	8
8,0	0	0	150.122992736	0	D	R	381786048	+	8
8,0	0	0	150.122992736	0	D	R	381786056	+	8
8,0	0	0	150.122992736	0	D	R	381786064	+	8
8,0	0	0	150.122992736	0	D	R	381786072	+	8

3.2.3. LSTM 학습 데이터 구성

페이지 해시 테이블 생성

LSTM 학습 데이터는 원본 트레이스 파일의 방대한 I/O 요청을 페이지별 접근 패턴 정보로 효율적으로 변환하는 과정을 통해 생성한다. 이 과정의 핵심은 실시간으로 각 페이지의 누적 통계치를 기록하는 해시 테이블을 구축하는 것이다.

이 해시 테이블은 논리 페이지 번호(LPN)를 고유한 키로 사용하며, 각 키에 해당하는 값으로는 해당 페이지의 접근 패턴을 요약한 누적 통계치들을 딕셔너리 형태로 저장한다. 페이지에 대한 정보가 해시 테이블에 존재하지 않을 경우, 첫 접근 시 자동으로 새로운 항목이 생성되고 초기화된다. 이렇게 해시 테이블이 초기화되면 다음 통계치를 기록한다.

- access_count: 해당 페이지에 대한 총 쓰기 접근 횟수를 기록한다.
- time_gap_sum: 모든 접근 간 시간 간격의 총합을 누적한다.
- time_gap_sq_sum: 표준편차 계산을 위해 시간 간격 제곱의 총합을 기록한다.
- request_size_sum: 해당 페이지에 대한 모든 쓰기 요청 크기의 총합을 저장한다.
- last_access_time: 가장 최근 쓰기 요청이 발생한 시점의 타임스탬프를 갱신한다.
- last_lifetime: 직전 접근 시점과 현재 시점 사이의 시간 간격을 계산하여 기록한다.
- history_features: 각 접근 시점마다 위 누적 통계치들을 기반으로 계산된 4가지 특징 스냅샷을 순서대로 저장하는 deque다. deque는 LSTM 시퀀스 길이(sequence_length)에 맞춰 가장 최근의 기록만을 효율적으로 유지하며, 새로운 데이터가 추가될 때 가장 오래된 데이터는 자동으로 삭제되어 메모리 오버헤드를 방지한다.

트레이스 파일을 처음부터 끝까지 한 줄씩 읽으면서, 각 쓰기 요청이 발생할 때마다 해당 LPN의 정보를 해시 테이블에서 찾는다. 이미 존재하는 페이지라면 기존 통계치에 현재 요청의 정보(타임스탬프, 요청 크기 등)를 더해 갱신한다. 이후, 통계적 의미를 갖기에 충분한 데이터가 확보되지 않은 페이지들, 즉 접근 횟수가 2회 이하인 페이지들은 LSTM 학습 데이터 구성에서 제외한다. 이 과정을 통해 해시 테이블은 방대한 I/O 트레이스 스트림을 각 페이지별로 압축된 형태의 접근 기록으로 변환하며, 이는 LSTM 모델이 과거의 접근 패턴을 인식하고 미래를 예측하는 데 필요한 기초 데이터를 제공하는 핵심적인 역할을 한다.

네 가지 통계 특성(feature) 추출

해시 테이블 작성을 완료한 후, 접근 횟수가 2 이하인 페이지(노이즈 페이지)는 통계 정보가 부족하므로 미리 제외한다. 남은 각 페이지에 대해 다음 네 가지 통계치를 계산해 4차원 피처 벡터를 구성한다. 이 특징들은 LSTM 모델이 페이지의 접근 패턴을 학습하는 데 사용되는 핵심적인 요소들이다.

- **시간 간격 평균 (Mean Time-gap)**
  페이지에 대한 쓰기 요청 간 평균적인 시간 간격을 나타내는 지표다. 이는 데이터의 접근 주기성을 파악하는 데 매우 중요하다. 만약 시간 간격 평균이 작다면 해당 페이지가 짧은 시간 내에 여러 번 덮어쓰기되는 'Hot' 데이터일 가능성이 높고, 이와 반대로 시간 간격 평균이 크다면 'Cold' 데이터일 가능성이 높다. LSTM 모델은 이 특징을 통해 페이지가 얼마나 자주 접근되는지를 학습하여 Hot/Cold 데이터를 분류하는 데 활용한다.
- **시간 간격 표준편차 (Standard Deviation of Time-gap)**
  쓰기 요청 간 시간 간격의 변동성을 보여주는 지표다. 표준편차가 낮을수록 접근 주기가 일정하고 예측 가능한 패턴을 가지며, 이는 주기적인 시스템 로그나 메타데이터와 같은 데이터에 해당된다. 반대로 표준편차가 높다면 접근 주기가 매우 불규칙하다는 것을 의미하며, 이러한 불규칙성은 예측이 어려운 워크로드의 특성을 나타낸다. LSTM 모델은 이 특징을 통해 시간 간격의 일관성을 파악하고 패턴의 규칙성 또는 불규칙성을 학습한다.
- **요청 크기 평균 (Mean Request Size)**
  해당 페이지에 대한 평균 쓰기 요청 크기를 나타낸다. 이 특징은 특정 응용 프로그램의 워크로드 특성을 반영하는 경우가 많다. 예를 들어, 작은 단위의 빈번한 쓰기 요청은 데이터베이스 로그와 관련될 수 있고, 큰 단위의 쓰기 요청은 멀티미디어 파일이나 백업 작업과 관련될 수 있다. LSTM 모델은 이 특징을 통해 페이지가 어떤 종류의 워크로드에 의해 접근되는지를 간접적으로 학습할 수 있다.
- **마지막 수명 (Last Lifetime)**
  가장 최근 접근 시점으로부터 현재 시점까지의 시간 간격이다. 이는 페이지의 '현재' 상태를 반영하는 가장 중요한 지표다. 페이지가 최근에 다시 접근되었다면 이 값은 작을 것이고, 오랫동안 접근되지 않았다면 큰 값을 가질 것이다. 이 지표는 LSTM 모델이 페이지의 최신 접근성을 기준으로 Hot/Cold 상태를 판단하는 데 핵심적인 역할을 한다.

3.2.4. K-means 군집화 및 LSTM 입력 시퀀스 생성

K-means 군집화 (K=4)

LSTM 모델의 예측 대상인 Hotness 레이블은 4차원 특징 벡터를 이용한 비지도 학습 알고리즘인 K-means 클러스터링을 통해 정의된다. 클러스터링에 앞서, 각기 다른 단위와 값의 범위를 가진 특징들의 스케일을 맞춰주는 정규화 과정을 수행한다. 이 과정은 데이터의 편향을 없애고 모든 특징이 동등한 중요도를 갖도록 하여, 모델이 데이터의 본질적인 패턴을 더 정확하게 학습하도록 돕는다.

이후, K-means 알고리즘을 적용하여 데이터를 네 개의 군집으로 분류한다. 이러한 다단계 분류는 FTL의 블록 할당 및 GC 정책 결정 시 더 정밀한 제어를 가능하게 한다. 클러스터링 결과, 각 페이지는 접근 패턴의 유사성에 따라 네 가지 Hotness 레이블을 부여받게 되며, 이 Hotness 레이블은 LSTM 모델의 지도학습(Supervised Learning)을 위한 정답 값으로 사용된다.

LSTM 입력 시퀀스 생성

LSTM 모델은 순차 데이터의 패턴을 학습하는 데 특화되어 있으므로, 앞서 생성한 페이지별 특징 벡터를 시간 순서에 맞는 시퀀스 형태로 재구성하는 과정이 필요하다. 본 연구에서는 과거 2개의 연속된 특징 벡터를 한 묶음으로 하는 슬라이딩 윈도우 기법을 적용했다.

이 방식은 각 페이지의 history_features 리스트에서 연속된 두 개의 특징 벡터를 '과거' 정보로, 바로 다음에 오는 특징 벡터를 '미래' 정보로 활용한다.

- **시퀀스 입력(X)**: 길이가 2인 슬라이딩 윈도우에 포함된 두 개의 특징 벡터를 묶어 LSTM의 입력 시퀀스를 구성한다. 예를 들어, deque에 [f1, f2, f3]가 저장되어 있을 때, 시퀀스 입력은 [f1, f2]가 된다.
- **정답 레이블(y)**: 시퀀스 입력(X) 바로 다음 시점의 특징 벡터(예: f3)에 해당하는 K-means 클러스터링 결과인 Hotness 레이블로 정의한다.

이러한 과정을 통해 (샘플 수, 시퀀스 길이, 특징 차원)의 3차원 배열 형태인 (N, 2, 4) 크기의 LSTM 학습 데이터를 생성한다. 이 데이터를 Stacked-LSTM 모델에 입력하여 과거의 접근 패턴으로부터 미래의 Hotness를 예측하게 된다.

3.3. Stacked-LSTM 모델 설계

3.3.1. 모델 구조 및 특징

Stacked-LSTM 모델을 채택한 것은 페이지별 접근 패턴의 시계열적 특성을 학습하기 위함이다. 이는 단일 LSTM 레이어보다 더 복잡하고 추상적인 데이터 패턴을 학습하는 데 유리하며, 모델의 표현력을 높여 과소적합(underfitting)을 방지하는 효과가 있다. 모델의 전체적인 구조는 4가지 통계 특징으로 구성된 4차원의 입력 데이터를 받아, 4개의 Hotness 클래스로 분류하는 다중 분류 모델로 설계되었다.[4]

모델은 입력 레이어부터 출력 레이어까지 다음과 같은 구조로 구성된다.

[표 6] Stacked-LSTM 모델 구조

레이어	입력 차원	출력 차원	주요기능
입력	(N, 2, 4)	(N, 2, 4)	4가지 특징으로 구성된 시퀀스 데이터 입력
LSTM 1	4	64	시퀀스 데이터의 특징 학습
LSTM 2	64	16	첫 번째 LSTM 레이어의 출력을 받아 추가 학습
드롭아웃	16	16	특정 노드에 대한 의존성을 줄여 과적합 방지
완전 연결(FC)	16	4	최종적으로 4개 클래스 중 하나를 예측

여기서 N은 배치(batch) 내 데이터 샘플의 수를 의미하며, '2'는 과거 2개 시점의 시퀀스 길이, '4'는 각 시점의 특징(feature) 개수를 나타낸다.

3.3.2. 모델 학습 및 검증

모델 학습은 파이썬의 PyTorch[3] 라이브러리를 사용해 구현되었다. 학습에 사용된 주요 파라미터로는 배치 크기(Batch Size) 256, 에폭(Epochs) 100, 학습률(Learning Rate) 0.001 등이 있다. 모델의 예측이 정답과 얼마나 차이가 나는지 측정하는 손실 함수(Loss Function)로는 모델이 출력한 각 클래스별 확률과 실제 정답을 비교하여 오차를 계산하는 방식을 사용했다. 이 과정에서 Hotness 클래스별로 데이터 수가 불균형한 문제를 해결하기 위해, 데이터가 적은 클래스의 오차에 더 큰 가중치를 부여하여 모델이 특정 클래스에 편향되지 않도록 보정했다. 모델의 가중치를 최적화하여 손실을 최소화하는 최적화기(Optimizer)를 사용했다.

또한, 학습 효율성과 안정성을 높이기 위해 두 가지 기법을 적용했다. 혼합 정밀도(AMP, Automatic Mixed Precision)를 활용하여 학습 속도를 향상시키고 GPU 메모리 사용량을

줄였다. 그리고 검증 데이터의 손실(Validation Loss)이 10 에폭 동안 개선되지 않으면 학습을 자동으로 멈추는 조기 종료(Early Stopping) 기능을 구현하여 불필요한 학습 시간을 단축하고 과적합을 방지했다. 이렇게 학습된 모델은 검증 데이터셋을 통해 성능을 평가하고, 가장 우수한 성능을 보인 모델을 최종 모델로 저장한다.

3.4. 오프라인 레이블링 및 시뮬레이션 데이터 생성

학습이 완료된 Stacked-LSTM 모델을 활용하여 전체 트레이스 파일에 Hotness를 부여하고, 이를 시뮬레이션에 적용한다. 이 과정은 크게 오프라인 레이블링과 시뮬레이션 데이터 준비로 나뉜다. 이 방식은 시뮬레이션 시간을 크게 단축시키며, Hotness 예측 결과를 FTL 정책에 효과적으로 반영할 수 있다.

먼저, 오프라인 레이블링을 위해 페이지별로 분할된 트레이스 파일을 순차적으로 읽는다. 각 쓰기 요청에 대해서는 미리 학습된 모델과 StandardScaler를 이용해 Hotness를 예측한다. 표준화된 입력 시퀀스를 모델에 넣고, 예측된 Hotness 클래스(0~3)를 Hotness_Label 필드에 추가한다. 반면, 읽기 요청은 Hotness에 따른 데이터 변화를 일으키지 않으므로 Hotness_Label을 -1로 지정한다. 이 과정을 통해 Hotness 레이블이 추가된 모든 트레이스 레코드는 새로운 출력 파일로 저장된다.

이어서 시뮬레이션 데이터 준비 단계에서는, 오프라인 레이블링으로 생성된 페이지 단위의 트레이스 파일이 직접 시뮬레이터의 입력으로 사용된다. 이 과정에서 시뮬레이터는 Hotness 정보를 활용하여 페이지별 격리 저장을 수행함으로써 FTL의 GC 정책을 Hotness 기반으로 제어하게 된다.

이러한 과정을 통해 실제 I/O 워크로드에 부합하는 Hotness 정보가 시뮬레이션 데이터에 간편하게 삽입되며, 복잡한 딥러닝 모델을 시뮬레이터 실행 단계에서 실시간으로 적용하지 않고도 Hotness 예측 결과를 효과적으로 반영할 수 있다.

[표 7] 입력 데이터 예시 (페이지별 분할 트레이스)

Device	CPU	Seq	Timestamp	PID	Action	Op	Sector	+	Size
8,0	0	0	150.122992736	0	D	W	381785824	+	8
8,0	0	0	150.122992736	0	D	R	381785840	+	8

[표 8] 출력 데이터 예시 (레이블링된 트레이스)

Device	CPU	Seq	Timestamp	PID	Action	Op	Sector	+	Size	Hotness_Label
8,0	0	0	150.122992736	0	D	W	381785824	+	8	2
8,0	0	0	150.122992736	0	D	R	381785840	+	8	-1

3.5. SimpleSSD-SA 기반 성능 검증 환경 및 I/O 모델 분석

본 연구에서 제안하는 딥러닝 기반 FTL 정책의 성능을 정량적으로 검증하기 위해 채택한 시뮬레이션 환경은 오픈소스 SSD 시뮬레이터인 SimpleSSD-SA(Standalone)에 기반한다. SimpleSSD-SA는 실제 SSD의 복잡한 내부 동작을 여러 기능 계층으로 정교하게 모듈화하고, 모든 연산을 나노초(ns) 단위로 스케줄링하는 이벤트 기반(Event-driven) 아키텍처를 채택하여, 미시적인 수준에서의 정밀한 성능 분석을 가능하게 한다. 본 절에서는 연구에 사용된 트레이스 파일의 단일 I/O 요청이 시뮬레이터의 각 계층을 거치며 처리되는 전체 흐름을 기술하고, 이 과정에서 SSD 성능에 결정적인 영향을 미치는 핵심 개념들을 심층적으로 설명한다.

3.5.1. SimpleSSD-SA의 계층적 아키텍처

SimpleSSD-SA의 아키텍처는 실제 SSD 컨트롤러의 펌웨어 및 하드웨어 구조를 추상화한 계층적 모델로 구성된다. 각 계층은 독립적인 역할을 수행하며, I/O 요청은 최상위 계층부터 최하위 계층까지 순차적으로 전달되고 처리 결과가 다시 상위로 반환되는 구조를 가진다.

3.5.2. I/O 요청의 계층별 처리 흐름 분석

1. 트레이스 재생 및 I/O 요청 생성 (Trace Replayer)

역할: 시뮬레이션의 시작점으로, 실제 워크로드를 기록한 텍스트 기반 트레이스 파일을 읽어 들인다.

동작: Trace Replayer 모듈은 3.4절에서 생성된 오프라인 레이블링 트레이스 파일의 각 라인을 순차적으로 파싱(Parsing)한다. 각 라인은 타임스탬프(Timestamp), 연산 종류(읽기/쓰기), 논리 블록 주소(LBA), 요청 크기(Size), 그리고 본 연구에서 추가한 Hotness

예측 레이블(-1~3) 필드로 구성된다. 이 정보를 바탕으로 시뮬레이터 내부에서 통용되는 표준 I/O 요청 객체(BlockIO)를 생성하고, 트레이스의 타임스탬프에 맞춰 시뮬레이션의 시간 흐름을 관리하는 Engine에 해당 요청 처리 이벤트를 등록한다. 또한, 이 계층에서 호스트(애플리케이션) 관점의 총 논리 쓰기량(write.bytes)이 최초로 집계되어 쓰기 증폭률(Write Amplification Factor, WAF) 계산의 분모가 된다.

2. 내부 캐시 계층 (ICL: Internal Cache Layer)

역할: SSD 내부에 탑재된 고속 DRAM 버퍼 캐시의 동작을 묘사한다.

동작: TraceReplayer로부터 생성된 I/O 요청은 SSD 컨트롤러로 진입하여 가장 먼저 ICL로 전달된다. 쓰기 요청의 경우, ICL은 데이터를 내부 버퍼에 일시적으로 저장(Write Buffering)하여 즉각적인 응답을 보낸 뒤, 특정 캐시 정책(Cache Policy)과 에빅션(Eviction) 정책에 따라 여러 개의 작은 쓰기 요청을 하나의 큰 요청으로 병합하여 하위 계층인 FTL로 내려보낸다. 읽기 요청의 경우, 요청된 데이터가 캐시에 존재하면(Cache Hit) 낸드 플래시에 접근하지 않고 즉시 처리하여 응답 시간을 단축시킨다. ICL의 존재는 FTL이 인지하는 I/O의 크기, 순서, 빈도를 변화시키는 중요한 변수이다.

3. 논리-물리 주소 변환 계층 (FTL: Flash Translation Layer)

역할: SSD의 핵심 두뇌 역할을 수행하며, 호스트가 사용하는 연속적인 논리 주소를 낸드 플래시의 불연속적인 물리 주소로 변환하고 관리한다.

동작: ICL을 통과한 I/O 요청은 FTL에 도달한다. FTL은 Request 객체에 담겨온 Hotness 레이블을 확인하여, Hot 데이터는 Hot 데이터 전용 블록 풀(Pool)에, Cold 데이터는 Cold 데이터 전용 블록 풀에 쓰는 방식으로 데이터의 물리적 분리를 수행한다. 또한, FTL은 내부의 빈 블록(Free Block) 비율이 설정된 임계값(GCThreshold) 이하로 떨어지면, 2.1절에서 설명한 가비지 컬렉션(GC)을 트리거하여 공간을 확보하는 책임을 진다. FTL 계층에서는 ICL로부터 전달받은 쓰기량(host_write_bytes)과 실제 낸드 플래시로 내려보내는 물리 쓰기량(device_write_bytes)을 각각 추적하여 내부 쓰기 증폭(Internal WAF) 계산의 기반을 마련한다.

4. 병렬 처리 및 하드웨어 추상화 계층 (PAL: Parallelism Abstraction Layer)

역할: SSD의 물리적 하드웨어 구조와 동작을 추상화한다.

동작: FTL에서 변환된 물리 주소와 연산 정보는 최종적으로 PAL로 전달된다. PAL은 SSD의 물리적 구조인 채널(Channel), 다이(Die), 플레인(Plane) 등의 병렬 구조를 모사하며, FTL로부터 받은 쓰기/읽기/삭제 명령을 실제 낸드 플래시의 데이터시트에 명시된 동작 타이밍(e.g., tPROG, tRCMD, tERASE)에 맞춰 처리한다. 이를 통해 PAL은 각 하드웨어 자원의 사용률과 병목 현상을 시뮬레이션하고, 최종적인 I/O 처리 시간을 계산하여 상위 계층으로 반환한다.

3.5.3. 핵심 성능 변수 분석: 슈퍼페이지 병렬성과 WAF

1. 슈퍼페이지(Superpage)와 병렬성

SimpleSSD에서 슈퍼페이지는 SSD의 물리적 병렬성(채널, 다이, 플레인 등)을 최대한 활용하여 데이터 전송률을 높이기 위해, 여러 개의 기본 페이지를 논리적으로 묶어 하나의 단위처럼 동시에 접근하는 기술이다. 예를 들어, 4개의 채널을 가진 SSD에서 SuperblockSize를 'C'(Channel)로 설정하면, 4개의 각 채널에 있는 페이지 하나씩을 묶어 하나의 거대한 슈퍼페이지를 구성한다. FTL은 이 슈퍼페이지 단위로 데이터를 기록함으로써 단일 채널로 기록할 때보다 이론적으로 4배 빠른 쓰루풋을 달성할 수 있다.

그러나 이러한 병렬성의 이면에는 파편화(Fragmentation)라는 비용이 따른다. 특히 랜덤 쓰기 워크로드에서는 슈퍼페이지의 일부 슬라이스만 갱신되는 부분 쓰기(Partial Write)가 빈번하게 발생한다. 이 경우, 슈퍼페이지 전체가 유효한 데이터로 채워지지 않아 블록 내 공간 낭비가 발생하고, 추후 GC 시 복사해야 할 유효 페이지가 흩어져 있어 쓰기 증폭을 증가시키는 요인으로 작용할 수 있다.

2. 외부(External) WAF와 내부(Internal) WAF

본 연구에서 측정하는 WAF는 분석 관점에 따라 두 가지로 나뉜다.

- **내부 WAF (Internal WAF):** FTL의 효율성을 측정하는 지표로, FTL이 ICL로부터 받은 쓰기량 대비, FTL이 실제 낸드 플래시에 기록한 총 쓰기량의 비율이다 (device_write / host_write_at_FTL). 이 값은 주로 GC 과정에서 발생하는 오버헤드를 반영한다.
- **외부 WAF (External WAF):** 사용자(애플리케이션) 관점의 전체 시스템 효율성을 나타내는 지표로, 사용자가 최초로 요청한 쓰기량 대비, 낸드 플래시에 기록된 총 쓰기량의 비율이다 (device_write / application_write).

외부 WAF = (ICL 증폭) × (내부 FTL 증폭)

두 WAF가 다른 이유는 ICL의 존재 때문이다. ICL은 캐시 정책에 따라 여러 개의 작은 쓰기 요청을 하나의 큰 슈퍼페이지 단위로 병합하여 FTL로 전달할 수 있다. 이 과정에서 'ICL 증폭'이 발생하여 FTL이 받은 쓰기량(host_write_at_FTL)은 이미 애플리케이션이 요청한 원본 쓰기량보다 커질 수 있다. 따라서 내부 WAF가 1.0에 가깝더라도, ICL의 동작 방식으로 인해 외부 WAF는 1보다 훨씬 높은 값을 가질 수 있다. 본 연구는 FTL 단에서의 내부 WAF 개선에 초점을 맞추었으나, 전체 시스템의 성능을 종합적으로 이해하기 위해서는 이러한 계층 간 상호작용을 반드시 고려해야 한다.

3.6. 전계층 변경 요약(Hotness 전파·배치 체계)

3.6.1. 설계 목표: First-Touch 태깅·풀 분리로 WAF 최소화

본 연구는 각 쓰기 요청에 포함된 Hotness 레이블을 temperature라는 변수로 명명하여 블록을 첫 태깅(first-touch) 하는 방식을 사용한다. 쓰기 요청이 도착하면 해당 temperature에 따라 해당 온도 풀(Hot/Warm/Cold 등)에서 빈 블록을 하나 선택하고, 그 순간 그 블록을 해당 온도로 태깅한다. 태깅된 블록은 가득 찰 때까지 동일 온도의 데이터만 수용하며, 가득 차면 닫힘(closed) 상태가 된다. 이후 가비지 컬렉션(GC)에서 블록이 소거(erase) 되면 태그가 제거되어 free 블록으로 환원되고, 어떤 온도 풀에서든 재사용될 수 있다.

GC로 피해 블록을 선택하면, 그 안의 유효 서브페이지는 가능한 한 같은 온도 풀의 오픈 블록으로 우선 복사한다. 해당 풀에 적합한 오픈 블록이 없거나 가득 찬 경우에는 즉시 같은 풀에서 새 오픈 블록을 확보한 뒤 복사를 이어간다. 복사 단계에서도 풀 일관성을 지키면 Hot과 Cold의 혼입을 최소화하여 이후 GC에서 옮겨야 할 유효 페이지 수가 줄고, 결과적으로 내부 WAF와 지연이 감소한다. Hot 블록은 데이터가 빠르게 무효화되어 복사 부담이 작고 빠르게 회수·재사용되며, Cold 블록은 오랫동안 유효 상태를 유지해 불필요한 이동이 드물다.

이 설계는 온도 혼입을 원천적으로 차단해 블록 내 데이터의 시간적 특성을 균질화한다. 그 결과, Hot 블록은 GC 시 유효 페이지 복사량이 작아 오버헤드가 줄고, Cold 블록은 안정적으로 유지되어 전체 내부 WAF가 감소한다. 또한 temperature가 시간이 지나며

변하더라도 매 쓰기 시점에 새로운 온도 풀과 블록을 즉시 재선택하므로, 배치가 워크로드 변화에 동적으로 적응한다.

이러한 실험 목표를 달성하기 위해 오프라인 예측된 Hotness 레이블을 시뮬레이터 전 경로에서 손실 없이 전파하여, FTL 단계에서 레이블 기반 물리 배치와 정량 지표(WAF·레이블별 쓰기량) 산출이 일관되게 이뤄지도록 하였다. 이를 위해 3.5장에서의 처리 흐름을 따라 트레이스의 Hotness 필드를 표준화된 temperature로 통일하고, ICL→FTL 흐름에 레이블 보존/주입 로직을 추가하였다.

3.6.2. Trace Replayer: 레이블 파싱·주입

트레이스 한 줄에서 Hotness 필드를 안전하게 정수 파싱하여 linedata.temperature에 저장한다. 제출 시, 생성되는 BIO에 bio.temperature = linedata.temperature로 직접 탑재해 다음 계층으로 전달한다.

3.6.3. None 드라이버(sil/none): BIO→HIL 전달

BIO를 HIL::Request로 변환할 때 req.temperature = bio.temperature로 손실 없이 복사한다.

3.6.4. ICL 수정: 캐시 사이드카로 레이블 보존·Flush 재주입

핵심: ICL의 쓰기 버퍼링·병합·에빅션 과정에서 레이블이 소실되지 않도록, 캐시라인 단위 사이드카 메타데이터를 도입하고 flush 시 FTL 요청에 재주입한다. 이를 위하여 ICL과 ICL 내부의 Generic Cache를 수정했다.

1. 사이드카 저장

쓰기가 라인 크기 미만으로 들어와 라인이 더티가 되면, 해당 라인의 포인터를 키로 lineTemp에 req.temperature를 저장한다.

2. flush 시 재주입

더티 라인을 플러시할 때, 사이드카에 보관된 값을 reqInternal.temperature에 채워 넣어 FTL로 전달전달하고, 완료 후 사이드카 항목을 제거해 누수를 방지한다.

3. 정상 경로 영향 최소화

캐시 교체 정책·지연 모델은 변경하지 않는다. 무효화·포맷 등으로 라인이 제거될 때도

사이드카 메타데이터를 함께 정리하여 일관성을 유지한다.

3.6.5. Hotness page mapping을 위한 FTL 수정 : 레이블 정규화·풀 선택·일관 GC·WAF 산출

핵심: FTL 입구에서 req.temperature를 수신·정규화하고(0~3), 레이블별 풀(Pool)에서 free 블록을 선택해 물리적 분리 기록을 수행한다. 레이블별 통계와 WAF도 본 모듈에서 집계·노출한다.

1. 레이블 수신·정규화 및 LPN

writeInternal() 입구에서 req.temperature를 읽어 clampTemp함수를 통해 0~3 구간으로 정규화(reqT)하고, LPN 단위 힌트(lpnTemp)로 저장한다.

2. 레이블 기반 free 블록 선택과 실제 사용 풀 확정

우선 요청 레이블 풀(reqT)에서 I/O lane 라운드로빈 정책으로 오픈 블록을 선택한다. (getLastFreeBlock) 내부적으로 stripe 기반 분산과 첫-터치 태깅(first-touch tagging)을 통해 선택된 free 블록을 해당 풀로 표기한다.

3. 통계 누적 및 쓰기 수행

서브페이지 수를 기준으로 요청 레이블 분포(writesReq) 와 실제 사용 풀 분포(writesUse) 를 누적하고, 상이 시 fallback 매트릭에 기록한다. 이후 블록·페이지 인덱스를 할당하여 실제 쓰기를 수행하고, 필요 시 read-before-write를 처리한다.

4. GC 동작

트리거 조건은 실제 사용 풀의 free 블록 헤드룸 비율이 설정 임계치 이하로 하락할 때다. 피해 블록의 유효 서브페이지를 같은 풀의 열려 있는 오픈 블록으로 우선 복사하고, 가득 차면 즉시 새 오픈 블록으로 전환해 풀 일관성을 유지한다.

5. WAF 계산 및 노출

총 디바이스 프로그램 바이트와 GC 복사 서브페이지 수를 바탕으로 WAF를 산출하여 지표로 노출한다.

4. 연구 결과 분석 및 평가

4.1. K-means clustering 결과

딥러닝 모델 학습에 필요한 hotness 레이블은 K-means 클러스터링을 통해 생성되었다. 클러스터링은 페이지 접근 패턴을 종합적으로 나타내는 네 가지 통계적 특성(4차원 데이터)을 모두 사용해 진행되었다.

4.1.1. 데이터 개요

클러스터링과 모델 학습에는 총 42,106,156개의 I/O 요청 데이터가 활용되었다. 이 중 쓰기 요청이 64.14%로 높은 비중을 차지한다.

- 총 읽기 요청: 15,096,166개 (35.86%)
- 총 쓰기 요청: 27,000,990개 (64.14%)

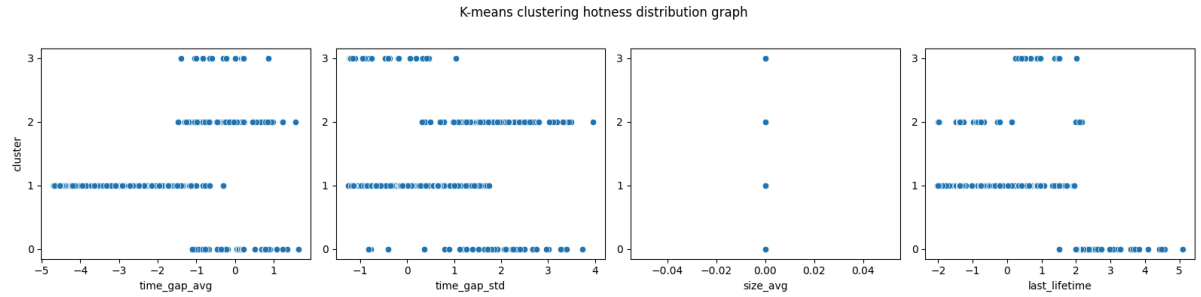
이 데이터를 기반으로 생성된 LSTM 시퀀스와 클러스터링용 다차원 데이터 수는 총 142,221개였다.

4.1.2. 클러스터링 결과

K-means 클러스터링을 통해 142,221개의 페이지 정보는 4개의 클러스터로 군집화되었다. 각 클러스터에 속한 페이지 정보의 분포는 다음과 같다.

- **Cold(0):** 16,841개 (11.8%)
- **Medium(1):** 42,259개 (29.7%)
- **Warm(2):** 52,322개 (36.8%)
- **Hot(3):** 30,799개 (21.7%)

클러스터링의 결과를 시각적으로 나타낸 K-means 클러스터링 Hotness 분포 그래프는 아래와 같다.



[그림 2] K-means 클러스터링 Hotness 분포 그래프

위 그래프는 네 가지 통계적 특성 각각에 대한 클러스터 분포를 보여준다. 왼쪽부터 순서대로 평균 시간 간격(time_gap_avg), 시간 간격 표준편차(time_gap_std), 요청 크기 평균(size_avg), 그리고 마지막 수명(last_lifetime)이다.

그래프에서 보듯이, 평균 시간 간격과 마지막 수명 특성은 클러스터별로 뚜렷한 패턴을 보이며 분포가 명확하게 분리되는 것을 확인할 수 있다. 이러한 분포 특성은 클러스터 번호가 페이지의 Hotness를 나타내는 유효한 지표임을 의미하며, 이 번호를 딥러닝 모델 학습을 위한 레이블로 사용했다. 그러나 요청 크기 평균 특성에서는 클러스터 간 명확한 분리가 나타나지 않았다. 이러한 현상에 대한 심층적인 분석 및 한계점은 4.5.2절에서 자세히 다룰 것이다.

4.2. LSTM 모델

4.2.1. 모델 구조 및 학습 파라미터

모델은 4가지 통계적 특성을 입력으로 받아 64개의 히든 차원을 갖는 첫 번째 LSTM 레이어와, 이어서 16개의 히든 차원을 가진 두 번째 LSTM 레이어를 거친다. 최종적으로 4개의 Hotness 클래스(Cold, Medium, Warm, Hot)를 분류하는 Dense 레이어로 구성된다. 전체 모델의 파라미터 수는 23,236개이다.

학습 과정에서 사용된 주요 파라미터는 다음과 같다.

- **배치 크기:** 256
- **Epochs:** 100
- **학습률(Learning Rate):** 0.001
- **활용 기술:** Automatic Mixed Precision(AMP), 드롭아웃(Dropout) 0.2, 조기 종료(Early Stopping) 10 에포크

4.2.2. 최종 성능 평가

학습이 완료된 후, 최적 성능을 보인 43번째 에포크의 모델을 테스트 데이터셋에 적용하여 최종 성능을 평가하였다.

```
=== 최종 테스트 평가 ===
최적 모델 로드 완료 (Epoch 43)
Test 결과: Loss=0.0253, Accuracy=99.81%
```

[그림 3] LSTM 모델의 최종 테스트 결과

모델은 99.81%의 높은 정확도(Accuracy)와 0.0253의 낮은 손실(Loss)을 기록했다. 이는 모델이 페이지의 Hotness를 거의 완벽하게 예측하는 뛰어난 성능을 보임을 증명한다.

모델의 예측 정확도를 시각적으로 보여주는 혼동 행렬은 다음과 같다.

```
=== 혼동 행렬 ===
[[ 6154     0     0     5]
 [    9 10449     0     4]
 [    0     0  3353    15]
 [    6     5    11  8434]]
```

[그림 4] Hotness 분류 모델의 혼동 행렬

혼동 행렬의 대각선에 위치한 값은 올바르게 분류된 샘플의 수를 나타내며, 이미지에서 볼 수 있듯이 오분류된 샘플의 수는 매우 적다. 이 결과는 Hotness 분류 모델이 매우 안정적이고 신뢰할 수 있음을 입증한다.

4.3. 시뮬레이션 설계 및 환경

본 장에서는 제안하는 딥러닝 기반 FTL 정책의 성능을 정량적으로 검증하기 위해 수행한 시뮬레이션의 환경, 결과, 그리고 심층 분석 내용을 기술한다.

4.3.1. 실험 목표 및 가설 검증

본 시뮬레이션의 최종 목표는 딥러닝으로 예측된 데이터 Hotness 정보를 FTL에 적용할 경우, 기존 정적 정책 대비 쓰기 증폭(WAF) 감소와 전반적인 I/O 성능 향상을 달성할 수 있는지 검증하는 것이다. 이를 위해 "딥러닝 기반 Hotness 분리 정책은 기본 페이지 매핑

정책보다 GC 효율을 극대화하여 WAF를 포함한 주요 성능 지표를 개선할 것"이라는 가설을 설정하였다.

4.3.2. 시뮬레이션 환경 구성

성능 비교를 위해 오픈소스 SSD 시뮬레이터인 SimpleSSD-SA를 활용하였다. 시뮬레이터의 주요 파라미터는 256GB 용량, 30%의 OP(Over-Provisioning), 그리고 64%의 GC Threshold로 설정하여 두 정책에 동일하게 적용하였다. 워크로드로는 SNIA IOTTA 리포지터리의 YCSB 블록 트레이스를 사용하였다.

비교 대상 1: 기본 페이지 매핑 (Baseline)

SimpleSSD-SA의 표준 페이지 레벨 매핑 FTL로, 데이터의 특성을 고려하지 않는 정적 정책이다.

비교 대상 2: 딥러닝 기반 Hotness 정책 (제안 기법)

사전 학습된 LSTM 모델이 트레이스의 쓰기 요청에 Hotness(0~3)를 예측하여 레이블링한다. FTL은 이 정보를 받아 Hot 데이터와 Cold 데이터를 물리적으로 다른 블록 그룹에 분리 저장하고, GC 수행 시 Cold 데이터가 많은 블록을 Victim으로 우선 선정하는 동적 정책이다.

4.4. 시뮬레이션 결과

동일한 환경에서 두 FTL 정책을 실행한 결과, 주요 성능 지표에서 아래와 같이 명확한 차이가 나타났다.

[표 9] 두 FTL 정책의 성능 지표 비교

지표 (단위)	페이지매핑	Hotness 기반페이지매핑	개선율
WAF (쓰기 증폭률)	136.39	1.03	약 99.2% 감소

IOPS (초당 처리량)	4,891	58,881	약 1103% 증가
평균 응답 시간 (ns)	380,967	6,980	약 98.2% 감소
GC 횟수	24,394	212	약 99.1% 감소

4.5. 결과 분석 및 고찰

4.5.1. 딥러닝 기반 정책의 성능 우수성 분석

실험 결과, 제안하는 딥러닝 기반 Hotness 정책은 WAF를 136.39에서 1.03으로 약 99.2% 감소시키는 압도적인 성능 개선을 보였다. 이는 Hot/Cold 데이터 분리 정책이 GC 효율을 극대화했기 때문이다.

기본 정책에서는 접근 빈도가 다른 데이터가 동일 블록에 혼재하여, GC 시 잦은 갱신이 없는 유효한 Cold 페이지를 불필하게 복사하는 오버헤드가 막대하게 발생했다. 이로 인해 GC 과정에서 발생하는 추가 쓰기(device_gc_write)가 WAF를 비정상적으로 증폭시키는 주원인이 되었다.

반면, 제안 기법은 잦은 덮어쓰기가 예상되는 Hot 데이터를 특정 블록에 격리함으로써 해당 블록이 빠르게 완전한 무효 상태가 되도록 유도했다. 그 결과, GC 시 유효 페이지 복사량이 획기적으로 줄어 WAF를 이상적인 수준인 1에 가깝게 유지할 수 있었다.

이러한 GC 효율 개선은 GC 횟수를 24,394회에서 212회로 대폭 감소시켰고, GC로 인한 성능 저하가 최소화되면서 IOPS는 1103% 증가하고 평균 응답 시간은 98.2% 단축되는 등 전반적인 I/O 성능의 폭발적인 향상으로 이어졌다.

한편, 기본 페이지 매핑의 WAF가 이처럼 높게 측정된 것은, 본 실험 환경의 GC Threshold(64%)가 GC를 강제로 유발하는 극단적인 설정이었기 때문이며, 이는 5.2절에서 연구의 한계로 상술한다.

4.5.2. 클러스터링 과정의 한계점

본 연구는 Hotness 레이블이 있는 데이터와 없는 데이터 간의 성능 비교에 집중하여 딥러닝 모델의 유효성을 검증하는 데 주력했다. 이를 위해 모델 학습에 평균 시간 간격, 시간 간격 표준편차, 요청 크기 평균, 마지막 수명의 네 가지 통계적 특성을 사용했다.

그러나 Hotness 레이블 비교라는 연구의 핵심 목표를 달성하기 위해, 원래 연구 설계에 포함되었던 페이지를 원래의 요청으로 재결합하는 과정이 생략되었다. 이 결정은 Hotness 레이블의 유무에 따른 모델의 성능 변화를 명확하게 비교하기 위한 방향성 조정의 일환이었다. 결과적으로 페이지 단위로 분할된 데이터가 독립적인 요청처럼 처리되었고, 모든 페이지의 요청 크기 값이 동일해지는 상황이 발생했다.

이러한 연구 진행 과정의 변경으로 인해 '요청 크기 평균'은 클러스터링 및 모델 학습 과정에서 더 이상 의미 있는 정보를 제공하지 못했다. 이는 4.1.2절에서 제시된 클러스터링 결과를 시각화한 그래프에서도 확인할 수 있다. '요청 크기 평균' 특성은 다른 세 가지 특성처럼 클러스터별로 명확하게 분리되는 패턴을 보이지 않았다. 이는 '요청 크기 평균'이 Hotness 분류에 실질적인 기여를 하지 못했음을 의미한다.

결론적으로, 본 연구는 의도치 않게 나머지 세 가지 특성만으로 모델 학습 및 Hotness 분류가 이루어진 한계를 가진다. 이는 연구 방향의 우선순위 설정으로 인해 발생한 필연적인 결과이며, 향후 연구에서는 모든 특성이 Hotness 예측에 효과적으로 기여할 수 있도록 데이터 전처리 과정을 보완할 필요가 있다.

4.5.3. 산업체 자문 의견에 대한 검토 및 답변

본 실험 결과는 산업체 자문 과정에서 제시된 주요 질의에 대한 실증적 답변을 제공한다.

WAF 외 성능 지표 제시: 자문위원은 WAF 외 쓰기 지연 시간 및 수명에 대한 가시적 데이터 추가를 제안했다. 본 실험에서 **평균 응답 시간(Latency)이 98.2% 단축**되었음을 확인함으로써 쓰기 지연 시간 개선을 명확히 입증했다. 또한, WAF가 136배에서 1.03배로 감소한 것은 NAND 플래시의 총 쓰기량이 획기적으로 줄었음을 의미하며, 이는 P/E 사이클 소모 절감을 통한 SSD 수명 연장 효과로 직결된다는 강력한 정량적 근거가 된다.

GC Victim 블록 선택 방식의 차별성: 제안 기법이 실제로 다르게 Victim 블록을 선택하는지에 대한 질의가 있었다. GC 횟수가 2만 4천여 회에서 212회로 감소한 결과는 두 정책의 Victim 블록 선택 전략이 근본적으로 다르게 동작했음을 명백히 보여준다. 제안

기법은 Hotness 예측을 통해 '좋은' Victim(무효 페이지가 많은 블록)을 효율적으로 식별하고 처리한 반면, 기본 정책은 그렇지 못해 잦고 비효율적인 GC를 유발한 것이다.

Cold 블록 GC 정책 및 Wear-Leveling: "Cold 페이지가 많은 블록은 유효 페이지 또한 많아 GC 오버헤드가 클 수 있고, 이로 인해 Wear-Leveling이 저해될 수 있다"는 우려가 제기되었다. Hot/Cold 데이터 분리 정책이 특정 블록의 마모를 가속화하여 블록 간 마모 불균형을 유발할 수 있으며, 이는 Wear-Leveling의 목표와 상충될 수 있다는 우려는 타당하다. 본 연구에서 제안하는 FTL 정책은 상충될 수 있는 두 가지 목표, 즉 GC 효율 극대화(WAF 감소)와 마모 균등 분배(Wear-Leveling) 중 전자에 우선순위를 둔 설계이다.

1. **핵심 목표: 총 삭제 횟수(Total Erase Count)의 절대적 감소** 제안된 정책의 핵심 원리는 Hot 데이터를 특정 블록에 집중시켜 해당 블록이 빠르게 무효화되도록 유도하는 것이다. 그 결과, GC 수행 시 복사할 유효 페이지가 거의 없어 GC 오버헤드가 최소화되며, 이는 SSD 전체에서 발생하는 불필요한 쓰기 및 삭제(Erase)의 총량을 획기적으로 줄이는 효과로 이어진다. 즉, 본 정책의 수명 향상 효과는 “마모의 균등 분배”가 아닌 “총 삭제 횟수의 절대적 감소” 에 기인한다.
2. **한계 및 보완 방안: Wear-Leveling 정책과의 통합** 본 연구의 시뮬레이션 범위는 Hot/Cold 분리 정책과 직접 연동되는 별도의 동적 Wear-Leveling 알고리즘의 구현을 포함하지 않는다. 따라서 장기적인 관점에서는 마모 불균형이 심화될 수 있는 한계를 가진다. 따라서 향후 연구 과제로는 현재의 Hot/Cold 분리 정책을 기반으로, 블록 간 마모도 편차를 주기적으로 모니터링하여 필요시 Cold 블록을 GC Victim으로 선택하는 지능형 Wear-Leveling 알고리즘을 통합하는 방안을 고려할 수 있다.

5. 결론 및 향후 연구 방향

5.1. 결론

본 연구는 딥러닝 기반의 Hotness 예측 정보를 FTL에 접목한 동적 데이터 관리 정책을 제안하고, 시뮬레이션을 통해 그 성능을 검증하였다. 실험 결과, 제안 기법은 기존 정적 페이지 매핑 방식에 비해 WAF, IOPS, 평균 응답 시간을 큰 폭으로 개선하며 SSD의 성능과 내구성을 동시에 향상시킬 수 있음을 성공적으로 입증하였다.

특히, WAF를 136.39에서 1.03으로 감소시킨 것은 제안한 Hot/Cold 데이터 분리 관리 기법이

GC 효율성을 극대화하고 불필요한 내부 쓰기를 최소화하는 데 매우 효과적임을 보여준다. 이러한 결과는 딥러닝 모델을 스토리지 시스템 내부 정책에 적용하는 연구의 실효성과 높은 잠재력을 확인하였다는 점에서 의의를 가진다.

5.2. 연구의 한계

본 연구는 긍정적인 성과에도 불구하고 다음과 같은 명백한 한계를 가진다.

첫째, **비교군(Baseline)의 실험 환경 설정 문제이다.** 기본 페이지 매핑 정책의 WAF가 136.39로 비정상적으로 높게 측정된 것은, 실험 환경에서 GC 발생을 의도적으로 유발하기 위해 GCThreshold를 64%라는 비현실적으로 높은 값으로 설정했기 때문이다. 이는 제안 기법의 성능 개선율을 실제 환경보다 과장되게 보일 수 있는 해석상의 한계를 가진다. 즉, 제안 기법의 우수성은 명확하나, 그 개선폭은 보다 현실적인 임계값(예: 15%~30%)을 적용한 환경에서 재검증될 필요가 있다.

둘째, **오프라인 학습 모델의 한계이다.** 본 연구에서 사용된 LSTM 모델은 사전에 수집된 트레이스를 기반으로 오프라인에서 학습되고, 예측된 Hotness 레이블을 정적으로 부여하는 방식이다. 이는 실시간으로 워크로드의 패턴이 변화하는 실제 컴퓨팅 환경에 능동적으로 대응하기 어렵다는 본질적인 한계를 내포한다.

셋째, **시뮬레이터 자체의 제약 사항이다.** 실험에 사용된 SimpleSSD-SA는 단일 스레드 기반으로 동작하며 Page-Level Mapping FTL 구조에 특화되어 있어, 병렬 GC나 하이브리드 매핑 등 최신 SSD의 복잡한 동작을 정밀하게 모사하는 데에는 제약이 따른다.

5.3. 향후 연구 방향

위에서 언급된 한계를 극복하고 본 연구를 발전시키기 위해 다음과 같은 후속 연구를 제안한다.

현실적인 Baseline 환경에서의 추가 검증: 보다 현실적인 GC 임계값(예: OP의 50~70% 수준)을 적용한 Baseline 환경을 구축하고, 해당 환경에서 제안 기법의 성능을 재검증하여 보다 객관적인 개선 효과를 측정하는 연구가 요구된다.

온라인 학습 모델 도입: 현재의 오프라인 학습 모델을 넘어, SSD 내부에서 실시간으로 워크로드 패턴 변화를 학습하고 정책을 동적으로 갱신하는 온라인 학습(Online Learning) 또는 강화 학습(Reinforcement Learning) 모델을 FTL에 통합하는 연구를 제안할 수 있다. 이는 실제 환경 변화에 유연하게 대응하는 차세대 지능형 FTL 개발의 기반이 될 것이다.

실환경 적용을 위한 확장 연구: 다양한 FTL 구조(Hybrid Mapping 등)를 지원하는 고도화된 시뮬레이터나, 실제 FPGA 기반 테스트베드에 제안 기법을 이식하여 상용 환경에서의 범용성과 실효성을 검증하는 연구 또한 요구된다.

클러스터링 피처 개선 및 동적 분류: 본 연구에서 '요청 크기 평균' 피처가 Hotness 분류에 효과적으로 기여하지 못했던 한계점을 보완하는 연구가 필요하다. 데이터 전처리 과정을 개선하여 페이지 단위로 분할된 데이터가 원본 요청의 크기 정보를 온전히 포함하도록 하고, 이를 통해 클러스터링 및 모델 학습의 정확도를 더욱 높일 수 있다. 나아가 K-means와 같은 정적 클러스터링 기법을 대체하여, 실시간으로 변하는 데이터의 특성을 반영할 수 있는 동적 분류 기법을 도입하는 연구도 제안한다.

6. 구성원별 역할 및 개발 일정

이준형	데이터 전처리 입출력 패턴 코드 분석 모델 개발
강인석	시뮬레이션 환경 구축 및 설정 성능 측정(WAF) Hotness 레벨 트레이스용 FTL 정책 개선 성능 비교
김지수	시뮬레이션 환경 구축 및 설정 FTL 개선 정책 구현 성능 측정 기존 FTL VS 딥러닝 기반 FTL 비교

업무	4				5				6				7				8				9			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
기획 및 설계																								
데이터 전처리																								
입출력 패턴 코드 분석																								
모델 개발																								
시뮬레이션 환경 구축 및 설정																								
FTL 개선 정책 구현																								
성능 측정																								

7. 참고 문헌

[1] SimpleSSD. "SimpleSSD: Open-Source Licensed Full-System SSD Simulator" [Online]. Available: https://docs.simplessd.org/en/v2.0.12/ (downloaded Sep. 9, 2025).

[2] SNIA. "IOTTA Repository: Input/Output Traces, Tools, and Analysis" [Online]. Available: https://iotta.snia.org/ (downloaded Sep. 9, 2025).

[3] PyTorch. "An open source machine learning platform that accelerates the path from research prototyping to production deployment." [Online]. Available: https://pytorch.org (downloaded Sep. 10, 2025).

[4] IBM Developer. "An introduction to deep learning with LSTM networks" [Online]. Available: https://developer.ibm.com/learningpaths/iot-anomaly-detection-deep-learning/intro-deep-learning-lstm-networks/ (downloaded Sep. 9, 2025)

[5] AGRAWAL, N., et al. "Design Tradeoffs for SSD Performance," Proc. of the 2008 USENIX Annual Technical Conference, pp. 1-14, Jun. 2008.

[6] YUNE, S. J., "A Study on Improving SSD Write Amplification through Machine Learning-based Hot/Cold Page Classification," Master's Thesis, Pusan National University, Busan, 2024.