**Pusan National University Computer Science and Engineering Technical Report 2023-10**

**딥러닝 기반 워크로드 분석을 통한**

**SSD 성능 개선**

**제목**



201724409 공희찬

201824476 박영훈

201824472 박선민

지도교수 안성용

목 차

[1. 서론 1](#_Toc148730339)

[1.1. 연구 배경 1](#_Toc148730340)

[1.2. 연구 목표 2](#_Toc148730341)

[2. 연구 배경 3](#_Toc148730342)

[2.1. 기존 연구 Hot/Cold 분류 3](#_Toc148730343)

[2.2. 기계학습 모델 3](#_Toc148730344)

[2.2.1. RNN(Recurrent Neural Network) 3](#_Toc148730345)

[2.2.2. LSTM(Long Short Term Memory) 4](#_Toc148730346)

[3. 연구 내용 5](#_Toc148730347)

[3.1. 데이터 가공 5](#_Toc148730348)

[3.1.1. 데이터 분석 5](#_Toc148730349)

[3.1.2. 데이터 전처리 5](#_Toc148730350)

[3.2. 레이블링 7](#_Toc148730351)

[3.2.1. 군집화 방법 선택 7](#_Toc148730352)

[3.2.2. 군집화 결과 분석 8](#_Toc148730353)

[3.3. 머신러닝 모델 설계 및 학습 10](#_Toc148730354)

[3.3.1. 모델 구조 10](#_Toc148730355)

[3.3.2. 모델 구현 11](#_Toc148730356)

[3.4. 시뮬레이터 11](#_Toc148730357)

[3.4.1. 시뮬레이터 설계 11](#_Toc148730358)

[3.4.2. 시뮬레이터 구현 12](#_Toc148730359)

[4. 연구 결과 분석 및 평가 15](#_Toc148730360)

[4.1. 연구 결과 15](#_Toc148730361)

[4.1.1. 모델 튜닝 15](#_Toc148730362)

[4.1.2. 시뮬레이터 결과 15](#_Toc148730363)

[5. 결론 및 향후 연구 방향 17](#_Toc148730364)

[5.1. 결론 17](#_Toc148730365)

[5.2. 한계 및 향후 연구 방향 18](#_Toc148730366)

[6. 구성원별 역할 및 개발 일정 19](#_Toc148730367)

[6.1. 구성원별 역할 19](#_Toc148730368)

[6.2. 개발 일정 20](#_Toc148730369)

[7. 참고 문헌 20](#_Toc148730370)

# 서론

## 연구 배경

SSD(Solid State Drive)는 플래시 메모리를 기반으로 한 비휘발성 저장 장치로 HDD(Hard Disk Drive)에 비해 높은 I/O 처리량, 빠른 입출력, 저전력, 저소음, 경량성, 낮은 지연시간 등의 장점 덕분에 널리 사용되고 있다. 하지만 SSD는 NAND 플래시 메모리로 구성되어 있어 데이터에 대한 덮어쓰기가 불가하며 쓰기와 삭제의 단위가 다르기 때문에 성능 저하가 발생할 수 있다. SSD의 페이지는 덮어쓰기 될 수 없는 특성을 가지고 있기 때문에, 데이터를 수정하기 위해서는 쓰기 전에 삭제(Erase Before Write) 작업이 필수적이다. 이 때, 읽기와 쓰기는 페이지(Page) 단위로 실행되며, 삭제는 블록(Block) 단위로 실행되기 때문에 문제가 발생한다. 한 페이지의 크기는 보통 4KB이고, 하나의 블록은 주로 128개의 페이지를 가지고 있어 512KB로 구성된다. 하나의 블록을 지운다는 것은 128개의 페이지를 삭제하는 것이다. 만약 이미 쓰기가 수행된 페이지에 새로운 쓰기가 수행된다면, 기존 페이지는 Invalid 상태가 되고, 쓰기는 블록 내 비어있는 페이지에 진행된다. 이 때 비어있는 페이지의 수가 임계치(Garbage Collection Trigger) 이하로 적어졌을 때, 가비지 컬렉션(Garbage Collection)이 수행된다. 가비지 컬렉션은 블록에 있는 모든 Valid 데이터들을 다른 비어있는 블록에 쓰고, 블록의 모든 페이지를 지워 해당 블록을 Empty 상태로 만드는 과정이다. 다른 비어있는 블록에 쓰는 과정에서 추가적인 쓰기가 발생되고, 이는 추가적인 수행 시간을 요한다. 나아가 플래시 메모리는 각 셀마다 쓰고 지우는 횟수(Program/Erase Cycle)가 정해져 있기 때문에, 해당 횟수를 모두 소모했다면 해당 셀은 수명을 다하여 더 이상 값을 저장할 수 없게 된다. 셀의 수명을 최대한 늘리기 위해 셀의 P/E Cycle 소모를 균등하게 하는 과정을 Wear Leveling이라고 하며, 이를 통해 추가적인 쓰기는 SSD의 성능과 수명에 악영향을 줌을 알 수 있다.

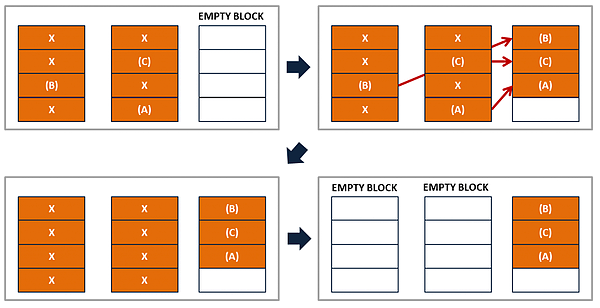


그림 1. 가비지 컬렉션 (Garbage Collection) [1]

## 연구 목표

SSD의 성능을 개선하기 위해서는 데이터의 입출력 패턴을 분석하는 것이 매우 중요하므로, 본 연구에서는 딥러닝 기법을 이용하여 데이터의 접근 패턴을 분석하고, 이를 이용해 SSD의 성능을 개선하는 기법을 연구하고자 한다. 데이터의 입출력 패턴을 분석하는 것이 중요한 이유는 다음과 같다.

첫 번째로, SSD는 NAND 플래시 메모리로 구성되어 있으며, 각 NAND 셀은 제한된 수의 P/E Cycle을 가지고 있다. Wear Leveling은 SSD의 수명을 최대화하기 위해 사용되는 기술로, 모든 NAND 셀이 균등하게 사용되도록 한다. SSD는 특정 영역에 지속적인 쓰기 작업이 집중되는 것을 원하지 않는다. 자주 데이터의 수정이 발생하는 Hot 데이터가 특정 영역에만 집중되면 그 영역의 NAND 셀이 빠르게 소모되고, 성능 저하가 발생된다. 데이터의 입출력 패턴을 파악함으로써, 자주 변경되는 Hot 데이터와 잘 바뀌지 않는 Cold 데이터를 분리하고, 이를 통해 SSD 내에서의 데이터 배치를 최적화할 수 있다. Cold 데이터와 Hot 데이터를 분리하여 저장하면, 데이터의 입출력 요청을 효율적으로 처리하고 SSD의 전반적인 성능을 향상시킬 수 있다.

두 번째로, SSD는 기존의 데이터를 직접 덮어쓸 수 없으므로, 새로운 데이터를 쓸 위치를 찾고, 기존의 데이터를 Invalid 상태로 만드는 Garbage Collection 프로세스를 거친다. Hot 데이터와 Cold 데이터의 입출력 패턴을 알면, Garbage Collection 작업을 최적화하여 불필요한 쓰기 연산을 최소화하고 성능을 향상시킬 수 있다.

결론적으로, SSD에서 데이터의 입출력 패턴을 분석하는 것은 SSD의 수명과 성능 측면에서 굉장히 중요하다. Hot 데이터와 Cold 데이터를 구분함으로써, SSD의 내부 동작과 자원 관리를 최적화할 수 있다.

위와 같은 이유들로, 우리는 우선 입출력 데이터를 Hot 데이터와 Cold 데이터로 분류할 것이다. 데이터를 분류하여 각 데이터를 최대한 서로 다른 블록으로 분리함으로써 전체 쓰기 증폭을 줄이며, GC가 좀 더 효율적으로 동작하도록 할 것이다. 필요에 따라 Hot 데이터와 Cold 데이터 두 종류로 구분 짓는 것뿐만 아니라, Warm 데이터와 Cool 데이터를 포함시키는 것이 더 효과적이라면 이들을 포함시켜 다양한 종류로 성능을 분석할 예정이다. 즉, 우리는 기계학습을 통하여 Hot 데이터와 Cold 데이터로 분류하는 기준을 세우고 입출력 워크로드 분석에 적합한 학습 모델을 개발할 것이다. 또한 SSD 시뮬레이터를 제작하여 모델의 학습을 이용한 SSD의 성능 개선을 확인하고 기존 SSD와의 성능을 비교할 것이다.

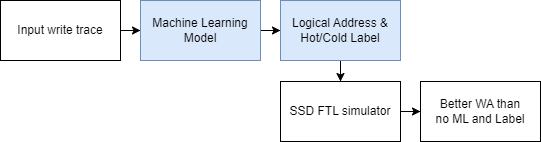


그림 2. 연구 목표 설계도

# 연구 배경

## 기존 연구 Hot/Cold 분류

SSD (Solid State Drive)의 성능향상을 도모하기 위해 Hot/Cold 데이터 분류를 사용하는 기존 연구 사례는 많이 존재한다. 이러한 연구들은 주로 SSD의 성능을 향상시키기 위해 데이터의 온도(데이터에 얼마나 자주 액세스되는지)에 따라 데이터를 분류하는 방법을 사용한다. 이를 통해 Hot 데이터는 빠른 응답 시간을 보이는 고속 SSD에 저장되고, Cold 데이터는 비교적 느린 저장장치에 저장되어 저장장치 용량을 효과적으로 확장할 수 있다.

## 기계학습 모델

### RNN(Recurrent Neural Network)

RNN은 이전 계층의 출력을 숨겨진 상태로 입력으로 사용할 수 있는 신경망이다. 예를 들어 영화 장면마다 어떤 상황이 벌어지는지 분류해야 한다. 이때 현재 어떤 상황이 일어나는 지를 이해하려면 이전에 어떤 상황이 일어났는지를 이해해야 한다. 이렇듯 이전 데이터의 정보가 현재 정보를 처리하는 데에 필요한 상황에 RNN이 사용된다.

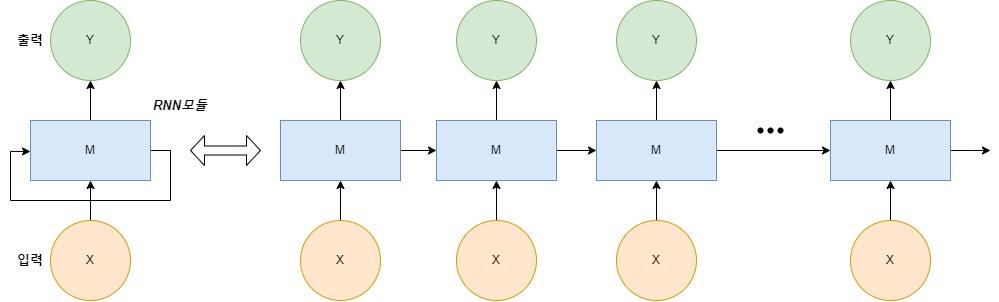


그림 3 . RNN 구조

### LSTM(Long Short Term Memory)

LSTM은 RNN의 일종으로 시계열 데이터에 대한 장기기억이 가능하다. RNN에서 발생하는 장기적 의존성 처리 문제를 해결하기 위해 제시되었다. 이러한 LSTM은 셀, 입력 게이트, 출 력 게이트, 망각 게이트로 구성된다. 이러한 세가지 게이트와 셀은 정보를 유지할지 망각할 지와 같은 정보의 흐름을 통제한다. 여기서 핵심적인 것은 셀 상태인데, 이는 아래 그림4에 서 상단에 위치한 수평 라인이다. 이러한 LSTM은 정보가 셀 라인을 따라 흐르면서 그 하단 에 위치한 게이트들을 이용하여 Hidden state 즉 이전 정보가 셀에 영향을 줄지 주지 않을 지를 결정한다. 이러한 게이트들은 sigmoid 신경망 레이어로 이루며 점별곱을 수행한다.

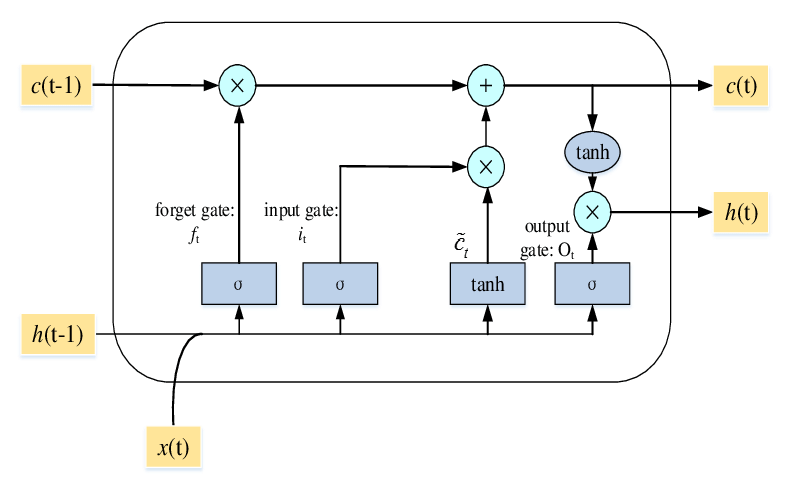


그림 4. LSTM 유닛 구조

# 연구 내용

## 데이터 가공

### 데이터 분석

실험에 사용할 데이터는 UMass Trace Repository의 OLTP Application I/O Trace[2] 데이터이다. 각 Trace Record는 단일 I/O command를 의미하며, 5개의 필드로 나누어진다. 데이터의 예시는 아래와 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Application Specific Unit | Logical Block Address | Size(byte) | Opcode | Timestamp(sec) |
| 1 | 240840 | 3072 | w | 0.026214 |

표 1. 전처리 수행 전의 I/O Trace

표 1의 데이터는 시작으로부터 0.026214초 지난 시점에, Application Specific Unit 1의 Logical Block Address 240840번지에 3072byte의 쓰기가 발생했다는 것을 의미한다. I/O Trace에서 Logical Block Address의 Block은 HDD(Hard Disk Drive)의 Sector를 의미하며, Sector 1개의 크기는 512byte이다. I/O Trace에서 Size의 단위는 byte이며, Timestamp의 단위는 second(초)이다. Application Specific Unit은 현재 진행하는 실험에 의미가 없으므로 제외할 필드이며, 실험의 주된 목적은 Write Amplication 감소이므로 Opcode가 “w” 즉, Write인 데이터만을 다룰 것이다.

### 데이터 전처리

우선 사용하지 않을 Application Specific Unit 필드와 Opcode필드를 제거하고, Logical Block Address를 Sector Number로 치환하여 Sector Number, Size, Timestamp 필드만 남겨두었다. Size 필드를 sector의 크기인 512 byte로 나누면 쓰기 시 차지하는 sector의 개수를 구할 수 있다. 위의 예시 데이터를 새롭게 변환한다면 다음과 같다.

|  |  |  |
| --- | --- | --- |
| Sector Number | # of Sectors | Timestamp(sec) |
| 240840 | 6 | 0.026214 |

표 2. Application Specific Unit, Opcode 필드를 제외하고 Size를 Sector 단위로 변환

이번 실험에서는 sector 단위가 아닌 페이지 단위로 레이블링을 진행할 예정이므로, sector를 페이지 단위로 환산해야 한다. sector 1개의 크기는 512byte이고, page 1개의 크기를 4KB로 설정하였으므로 하나의 page는 8개의 sector로 이루어져 있다. 표 2의 데이터는 Sector Number 240840번부터 240845번까지 6개의 sector에 I/O trace의 시작으로부터 0.026214초 후에 쓰기 작업이 실행되었다는 것을 의미한다. 이를 Page로 나타낸다면, Sector Number를 8로 나누고, # of Sectors도 8로 나누어 해당 I/O Trace가 몇 개의 페이지를 차지하는지 알 수 있다. 위 예시 데이터를 Page 단위로 변환하면 다음과 같다.

|  |  |  |
| --- | --- | --- |
| PageId | Size(byte) | Timestamp(sec) |
| 30105 | 3072 | 0.026214 |

표 3. Sector를 Page단위로 변환

우리는 이번 실험에서 군집화 진행 시 사용할 매개변수를 page의 접근 빈도(Frequency), page 접근시간 간격의 평균(Time Interval Average), page 접근시간 간격의 표준편차(Time Interval Std), size의 평균(Size Average)으로 설정하였다. 위의 전처리된 데이터에서 해당하는 매개변수들을 얻기 위해 우선 표3의 데이터들을 파이썬의 DataFrame으로 변환하였고, groupby 함수를 사용하여 Timestamp 순서로 정렬하고 각 집계값들을 계산하였다. 만약 해당 Page에 접근한 횟수가 1이라면 시간 간격은 존재하지 않을 것이므로, 접근시간 간격의 평균과 표준편차는 모든 집계값 중 최댓값으로 설정했다. 또한, Page에 접근한 횟수가 2라면 시간 간격은 1개 존재할 것이고 시간 간격의 표준편차는 존재하지 않기 때문에, 이 또한 최대값으로 치환하였다. 만약 최대값으로 치환하지 않고 0으로 치환한다면, 시간 간격의 표준편차가 매우 작다고 판단하여 Hot 데이터로 분류될 것이다.

최종적으로 전처리된 데이터는 아래와 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PageId | Frequency | Time Interval Average | Time Interval Std | Size Average |
| 30105 | 11785 | 1.1899 | 13.6513 | 3072.0434 |

표 4. 최종 전처리 데이터

표 4의 데이터는 Page 30105번지에서 11785회 쓰기가 진행되었으며, Page 접근시간 간격의 평균은 1.1899초이고 Page 접근시간 간격의 표준편차가 13.6513임을 의미한다.

## 레이블링

### 군집화 방법 선택

데이터가 자주 수정되면 Hot, 거의 수정되지 않으면 Cold로 구분하고 필요에 따라 그 사이에도 군집을 추가하여 데이터를 분류하기 위해 비지도 학습을 진행했다. 군집 수를 지정하지 않아도 되고, 이상치 대응 성능이 좋은 DBSCAN(Density-Based Spatial Clustering of Applications with Noise)를 군집화 도구로 고려했다. 하지만, 대용량 데이터를 다뤄야 하기 때문에 DBSCAN보다 단순한 k-means 알고리즘을 사용했다.

k-means의 매개변수로 페이지 접근 빈도, 접근 시간 간격 평균, 접근 시간 간격의 표준편차, 입력 크기의 평균을 사용했다. 짧은 시간에 집중적으로 수정된 후 긴 시간 동안 수정되지 않은 데이터의 경우, 접근 시간 간격의 평균만으로 구분할 수 없고, 접근 간격의 표준 편차를 추가 해야 했다.

데이터 분석 시, 변수들의 단위 차이로 인해 변수들의 특성을 무시하고 단순히 수치만 큰 값을 기준으로 분류되는 것을 피하기 위해 표준화를 수행하였다. 표준화 함수로 MinMaxScaler를 고려했지만, 이 함수는 데이터의 원래 분포를 유지하는 특성을 가지고 있어 접근 빈도가 낮은 쪽으로 데이터가 편중되어 있는 우리의 연구에 적합하지 않다고 판단했다.

MinMaxScaler 대신 사용한 StandardScaler의 경우 평균과 분산을 각 0과 1로 조정하고 분포 역시 변화하므로, 엘보우 기법으로 k값 4를 얻을 수 있었다.

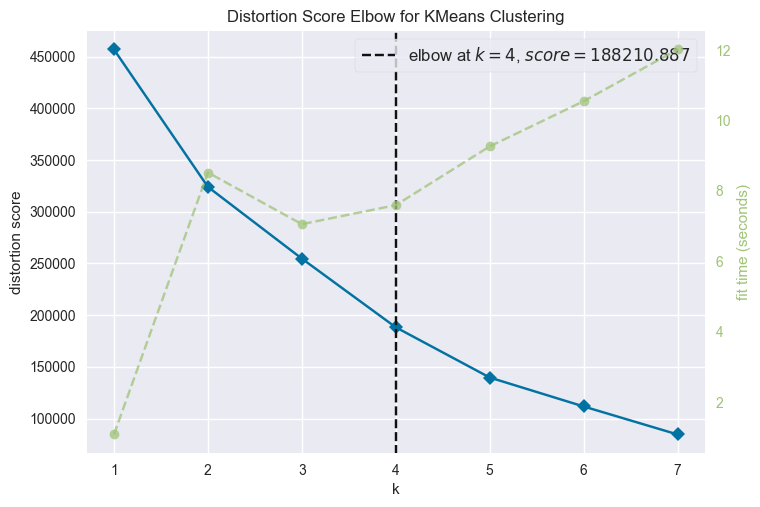


그림 5. 데이터 엘보우 기법 k 설정

### 군집화 결과 분석

데이터 특성 4개로 군집화 하면, 그래프를 그려 군집화를 시각적으로 확인하기 어렵다. 이에 PCA(Principal Component Analysis)를 활용하여 4차원에서 2차원으로 바꿔 아래처럼 그래프를 그려 군집화를 확인했다.

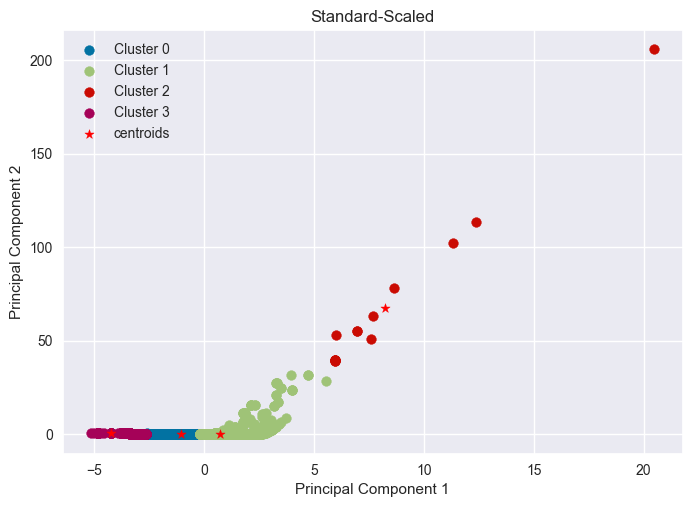


그림 6. PCA 활용 군집화 그래프

이제 각 군집이 Hot과 Cold 중 어디에 가까운 지 분석하기 위해 각 군집의 특성을 분석한다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Frequency** | **Time Interval-avg** | **Time Interval-std** | **Size-avg** | **Cluster** |
| 6 | 2385.969067 | 2937.106799 | 2816 | 0 |
| 4 | 2773.199585 | 2403.712900 | 3840 | 0 |
| 4 | 2773.225627 | 2403.747447 | 3840 | 0 |
| 104 | 104.177744 | 599.385970 | 512 | 1 |
| 178 | 61.051303 | 417.857584 | 512 | 1 |
| 128 | 84.295964 | 540.087874 | 512 | 1 |
| 3 | 370.760071 | 494.332637 | 4096 | 2 |
| 5 | 199.994568 | 347.688264 | 3379.2 | 2 |
| 4 | 266.499959 | 393.437277 | 4096.0 | 2 |
| 5941 | 2.360102 | 3.923239 | 1538.843966 | 3 |
| 5920 | 2.367561 | 3.957989 | 1537.816216 | 3 |
| 5938 | 2.360431 | 3.909648 | 1538.241832 | 3 |

표 5. 군집별 대표값의 특성

위의 특성값 분포를 토대로, Cluster 0을 Cold, Cluster 1과 3을 Hot, Cluster 2를 Warm으로 결정했다. 군집 3은 빈도가 극히 크기 때문에 Hot, 군집1은 빈도가 크고 Time Interval-avg가 작기 때문에 Hot이다. 군집 0은 Time Interval-avg가 극히 크기 때문에 Cold이다. 군집 2는 Hot Cold 둘 중 어느 것이라 하기에도 애매한 수치를 보이기에 Warm이다.

## 머신러닝 모델 설계 및 학습

### 모델 구조

이번 연구에 사용되는 lstm 모델의 구조는 아래와 같이 설계된다. 과적합을 막기 위해 Keras 에서 제공하는 L2 regularizer과 Dropout을 활용했다. 분류를 위해 softmax 함수 사용하여 Hot / Cold / Warm 3개 군집으로 분류했다. 최적화를 수행하는 옵티마이저에 따라 모델 학습의 안정성에 영향을 준다. 기울기 정도에 영향을 받는 옵티마이저의 사용은 실패의 가능성이 있다. 그러나 Adam의 경우 그러한 기울기 정도에 영향을 받지 않으므로 Adam을 모델의 옵티마이저로 사용했다.



그림 7. lstm 모델 구조

### 모델 구현

모델은 학습에 사용한 데이터는 전체 I/O Trace의 30%에 해당하며, 나머지 70%는 시뮬레이터의 동작에 사용하였다. 사전에 전처리를 진행했던 데이터의 형식은 [PageId, Frequency, Time Interval Average, Time Interval Std, Size Average]와 같으므로, 우리는 모델을 이용하여 입력 값(Feature)으로 [Frequency, Time Interval Average, Time Interval Std, Size Average]를 받아 Label (Hot/Cold/Warm)을 출력할 것이다. 입력 값 중 Frequency는 논리적 페이지 주소의 빈도를 나타내기 때문에 Frequency가 높을수록 Hot 데이터로 분류되며, Time Interval Average는 해당 논리적 페이지 주소의 접근 시간 간격의 평균을 나타내므로 작을수록 Hot 데이터로 분류될 것이다. Time Interval Std는 논리적 페이지 주소의 접근 시간 간격의 표준편차를 나타내며, 표준편차가 작을수록 주소에 접근을 균등하게 한다는 의미이므로 Hot 데이터로 분류된다. 논리적 페이지 주소에 쓰여지는 데이터의 크기가 클수록 데이터가 수정되는 현상이 적은 것을 고려하여, Size Average는 작을수록 Hot 데이터로 분류되도록 하였다.

## 시뮬레이터

### 시뮬레이터 설계

시뮬레이터는 FTL(Flash Translation Layer)의 동작을 모사한 것이다. FTL은 OS의 입출력 및 파일구조를 SSD에 사용하기 위한 논리 주소(Logical Address)와 물리 주소(Physical Address) 간의 매핑을 관리하여, 호스트 시스템이 논리 주소를 사용하여 데이터를 읽고 쓸 수 있도록 한다. 또한 Garbage Collection, Wear Leveling을 수행하는 역할을 하는데, 본 실험에서는 SSD의 수명보다 Write Amplication 감소로 인한 성능 향상에 집중한 실험이기에 Wear Leveling의 동작은 생략하였다. 제작한 시뮬레이터의 Flow Diagram은 아래와 같다.

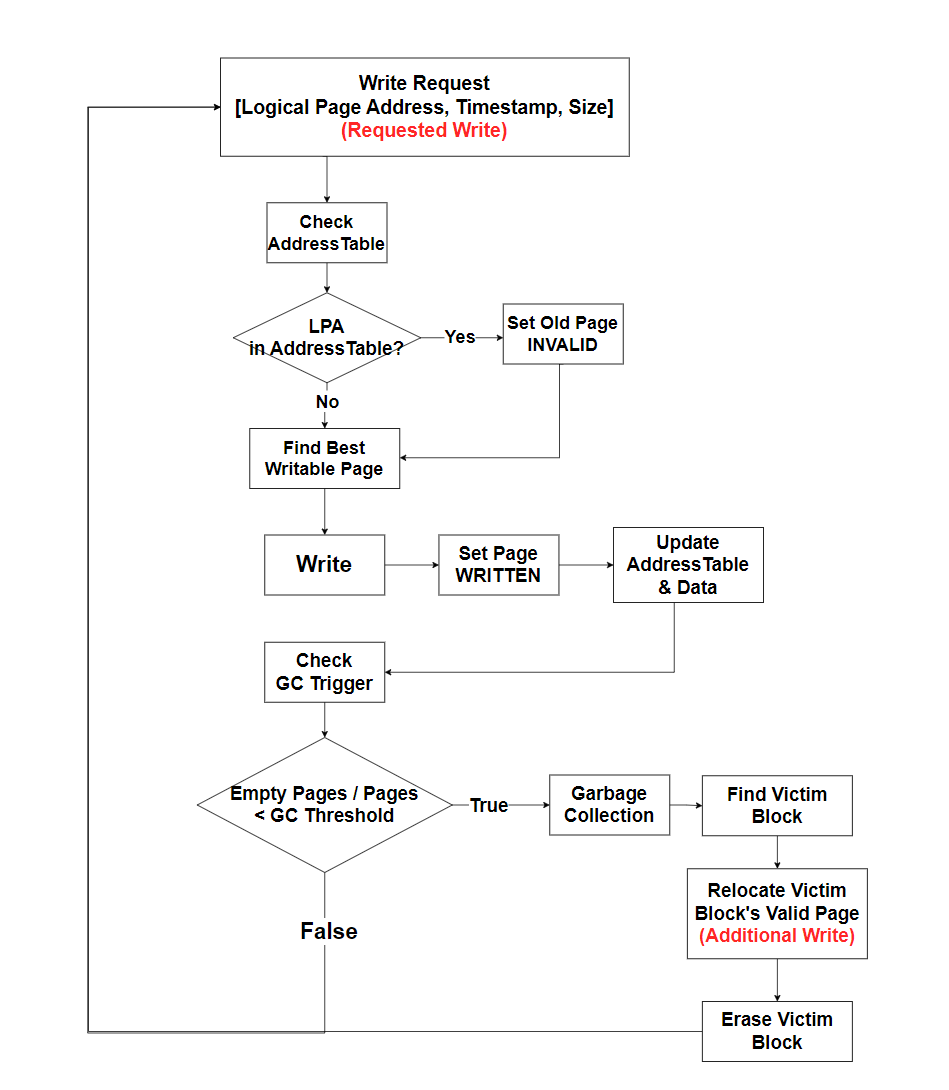


그림 8. 시뮬레이터 Flow Chart

### 시뮬레이터 구현

시뮬레이터는 Python을 이용한 객체 지향 프로그래밍으로 구현하여 SSD, Block, Page와의 관계와 소통을 중심으로 코드를 작성하였다. 사용 언어로 Python을 선택한 이유는 Tensorflow로 설계한 모델을 시뮬레이터에 손쉽게 적용할 수 있기 때문이다.

시뮬레이터는 두 가지 버전을 작성했다. 하나는 데이터의 Hot/Cold 구분없이 FTL(Flash Translation Layer)의 동작을 수행하는 방식으로 해당 버전의 WA(Write Amplification)가 비교 기준점으로 사용된다. 다른 시뮬레이터는 입력되는 I/O Trace에 대해 머신러닝으로 학습을 진행한 모델을 사용하여 실시간으로 Hot/Cold 여부를 판단하고, 이를 토대로 각기 다른 블록에 쓰기를 수행하는 버전이다. 이는 Cold 데이터들이 같은 블록에 모이도록 함으로써, 해당 블록 내 Invalid 상태인 페이지가 드물게 발생하여 블록 지우기의 대상이 되지 않도록 하여 결과적으로 WA를 낮추기 위함이다.

우리는 Hot/Cold 이진 분류보다는, Hot/Warm/Cold의 3가지 클래스로 분류하는 것이 적절하다고 판단하여, 시뮬레이터에 Hot/Warm/Cold 데이터를 적용할 수 있도록 설계했다. 이러한 테스트를 위해 시뮬레이터에 적용한 파라미터들은 다음과 같다. 한 페이지의 사이즈는 4KB이고, 하나의 블록은 128개의 페이지로 이루어져 있는 점을 고려하여, SSD에는 총 3000개의 블록이 있다고 가정하여 SSD의 크기는 약 1.5GB로 설정하고 실험을 진행했다. 이렇게 설정한 이유는 실험에 사용된 I/O Trace의 서로 다른 논리 페이지 주소가 약 15만개이기 때문에, SSD에서 충분히 수용가능하지만 용량이 많이 남지는 않도록 하게 설정하였다. 또 다른 파라미터로는 Garbage Collection이 발생하기 위한 Threshold와, SSD 내부 블록들의 Hot/Warm/Cold 레이블 비율이 있다. SSD 내부 블록의 Hot/Warm/Cold 비율을 설정함으로써, 각 분류된 데이터들이 어느 블록에 들어가야 할지에 대한 최소한의 가이드라인을 제공했다. 이와 관련된 동작은 3.4.2.2절에서 설명한다.

**3.4.2.1. 기계학습을 수행하지 않는 시뮬레이터 구현**

Hot/Cold 구분 없이 FTL의 동작을 모사하는 시뮬레이터가 동작하는 과정은 다음과 같다.첫 번째로 I/O Trace([SectorId, Size, Timestamp])가 입력으로 들어오는데 이를 Sector에서 Page단위로 변환한다. 이 때, Page는 논리적 주소이다. 그 다음 논리 주소를 SSD 물리 주소로 변환하기 위한 매핑 테이블을 확인하여 해당 논리 주소가 매핑 테이블에 존재하는지 확인한다. 논리 주소가 매핑 테이블에 존재한다는 의미는, 해당 주소는 이미 SSD에 쓰여져있기 때문에 Invalid 상태로 변경하고 다른 빈 페이지에 써야한다는 의미이다. 빈 페이지를 찾아 쓰는 과정은 모든 블록을 순회하며 Empty 페이지가 가장 많은 블록을 찾아 그 블록의 Empty 페이지에 쓰도록 설계했다. 만약 쓰기 동작이 수행되면, 그 페이지의 상태는 Written(Valid) 상태로 변경하고, 주소 매핑 테이블을 업데이트하며, 해당 페이지의 데이터를 업데이트한다. 특히 페이지에 데이터가 쓰여졌다면, GC Trigger를 확인하여 Garbage Collection이 발생해야 하는 지 확인해야 한다. Garbage Collection이 발생할 조건(GC Trigger)은 Write 가능한 전체 페이지의 수를 SSD 내부 전체 페이지의 수로 나눈 값이 Threshold보다 작을 때 발생하도록 설계했다. Garbage Collection이 동작하는 과정은 다음과 같다. 우선 지우기 동작을 수행할 블록(Victim Block)을 찾아야하는데, 이 블록은 모든 블록을 순회하며 Invalid 페이지가 가장 많은 블록을 선정하도록 구현했다. 그 다음 Victim Block의 모든 Valid 페이지를 다른 블록의 Empty 페이지에 쓴다. 이 과정에서 발생하는 추가적인 쓰기(Additional Write)가 WA 증가의 원인이다. 그 다음 Victim Block에 남아있는 모든 Invalid 페이지들을 지움으로써 블록을 빈 상태로 만든다. 이 모든 동작이 완료되면, 다음 I/O Trace를 입력으로 받는다.

**3.4.2.2. 기계학습을 수행하는 시뮬레이터 구현**

기계학습을 수행하는 시뮬레이터는 이를 수행하지 않는 시뮬레이터와 대부분 유사하게 동작하지만, 몇 가지 변경해야 하는 부분이 있다.

첫 번째로, [Logical Page Address, Timestamp, Size]를 입력값으로 받을 때 해당 LPA(Logical Page Address)의 특성값([Frequency, Time Interval Average, Time Interval Std, Size Average])을 실시간으로 업데이트 해야 한다. 이를 실시간으로 업데이트하는 이유는, 시간의 흐름에 따라 해당 LPA의 특성이 변화할 수 있음을 고려하였다. 예를 들어, 해당 LPA에 접근하는 시간 간격이 매우 짧고 접근 빈도가 매우 높아 Hot으로 분류되었다가, 시간이 흐름에 따라 점차 접근 시간 간격이 매우 길어지고 접근을 더 이상 하지 않는다면, Cold로 분류되어야 할 것이다.

두 번째로, 위에서 업데이트된 특성값들을 이용하여 기계학습을 진행하고, 해당 LPA에 대한 Labeling을 실시간으로 진행해야 한다. 이렇게 Labeling된 데이터는 주소 매핑 과정을 통해 SSD 내부의 일치하는 Labeling된 블록에 쓰여지게 될 것이다.

세 번째로, 쓰기 과정에서 Hot 데이터는 Hot 블록의 페이지에 쓰여질 것인데, 만약 모든 Hot 블록이 가득 찬 상태라면 Warm 블록의 페이지에 쓰여져야 할 것이고, 만약 Warm 블록까지 가득 찬 상태라면, Cold 블록의 페이지에 쓰여져야 할 것이다. 만약 Warm 데이터였다면 Hot 블록, Cold 블록 순으로 확인해야 할 것이며, Cold 데이터였다면 Warm 블록, Hot 블록 순으로 빈 페이지를 찾아 써야 할 것이다.

이 외의 다른 동작들은 기계학습을 수행하지 않는 시뮬레이터와 일치하도록 구현했다.

# 연구 결과 분석 및 평가

## 연구 결과

### 모델 튜닝

lstm unit size와 batch size를 2의 제곱수로 조정하며 모델 평가를 해봤다. lstm unit size는 64와 128에서 가장 좋은 성능을 보였고, batch size는 32와 64에서 가장 좋은 성능을 보였다.

class\_weight은 데이터 종류별로 개수가 달라 모델의 학습과 결과가 왜곡되는 현상을 막아주는 기능을 한다. 데이터 종류 비율의 역수를 class\_weight으로 넣어주면 된다. 학습에 쓰인 Hot : Cold : Warm 의 데이터 수의 비율은 29757 : 18546 : 65928 이다. 이 비율의 역수를 간단한 정수의 비로 나타내면 대략 2 : 3 : 1 이다.

### 시뮬레이터 결과

시뮬레이션에 적용한 파라미터는 페이지 크기, 블록 당 페이지 수, SSD 내부의 총 블록 수, Garbage Collection이 발생하는 Threshold, SSD 내부의 Hot: Warm: Cold 블록의 비율이다. 페이지 크기는 4KB, 블록 당 페이지 수는 128개, Garbage Collection Threshold 0.3으로 동일하게 고정하고, SSD 내부의 총 블록 수, 즉 SSD의 크기와 SSD 내부의 Hot: Warm: Cold 블록의 비율만 변경하며 시뮬레이션을 진행했다. 모델을 적용하여 시뮬레이션을 진행할 때 SSD 내부의 Hot:Warm:Cold 블록의 비율은 입출력 데이터에 따라 유동적으로 변해야 하는 것이 맞지만, 구현의 어려움으로 미리 그 비율을 정하고 시뮬레이션을 진행했다. 비율을 설정할 때는 시뮬레이션에 사용될 입출력 데이터의 패턴을 분석하여, Hot:Warm:Cold 비율을 결정하였다.

추가적으로 레이블링은 총 2가지 버전으로 진행되었는데, 하나는 Frequency를 포함한 레이블링과 다른 하나는 Frequency를 포함하지 않은 레이블링이다. 실험 초기에는 Frequency가 분명히 데이터의 분류에 영향을 끼칠 것이라고 판단하여 데이터의 특성에 포함시켰지만, 이후 특정 주소로의 접근이 초기에는 매우 빈번했다가, 이후에는 거의 접근하지 않게 되는 데이터의 특성이 변화할 수도 있는 가능성을 열어두고 Frequency를 데이터의 특성에서 제외한 버전으로도 레이블링을 진행하였다.

빈도를 포함하여 레이블링을 진행했을 때 시뮬레이션에 사용될 데이터의 Hot:Warm:Cold 비율은 10:1:2 였고, 빈도를 포함하지 않았을 때 데이터의 비율은 2:32:5였다. 블록의 온도 비율과 SSD의 크기를 변화하며 시뮬레이션을 진행한 결과는 다음과 같다.

|  |  |
| --- | --- |
| **파라미터** | **값** |
| 페이지 크기 | 4KB |
| 블록 당 페이지 수 | 128 |
| SSD 내부의 총 블록 수 | 2000 / 3000 |
| Garbage Collection Threshold | 0.3 |

표 6. 시뮬레이션에 사용된 파라미터

|  |  |  |  |
| --- | --- | --- | --- |
|  | Without Labeling | Hot/Warm/Cold with Frequency | Hot/Warm/Cold without Frequency |
| Requested Write | 3,939,772 | 3,939,772 | 3,939,772 |
| Additional Write | 4,267,975 | 4,097,218 | 4,144,108 |
| **Write Amplification** | **2.0833** | **2.03996** | **2.05186** |

표 7. 약 1GB SSD에서의 I/O Trace 394만 건에 대한 시뮬레이션 결과

빈도를 포함하여 레이블링을 진행 했을때, 약 2.1%의 Write Amplification 개선이 이루어졌음을 확인할 수 있고, 빈도를 포함하지 않고 레이블링을 진행했을 때는 약 1.5%의 Write Amplification 개선이 이루어졌음을 알 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Without Labeling | Hot/Warm/Cold with Frequency | Hot/Warm/Cold without Frequency |
| Requested Write | 3,939,772 | 3,939,772 | 3,939,772 |
| Additional Write | 1,401,428 | 1,392,176 | 1,426,435 |
| **Write Amplification** | **1.35571** | **1.35336** | **1.36206** |

표 8. 약 1.5GB SSD에서의 I/O Trace 394만 건에 대한 시뮬레이션 결과

빈도를 포함하여 레이블링을 진행 했을때, 약 0.2%의 Write Amplification 개선이 이루어졌음을 확인할 수 있었다. 빈도를 포함하지 않고 레이블링을 진행한 데이터에 대해서는 오히려 Write Amplification이 증가한 모습이다.

# 결론 및 향후 연구 방향

## 결론

SSD의 크기에 커짐에 따라 Write Amplification 값이 작아지는 이유는, 동일한 I/O Trace의 개수를 입력으로 받았을 때, 처음 Garbage Collection이 발동되는 시점이 늦어지고, 내부 메모리 공간을 더 효율적으로 사용할 수 있기 때문이다. 빈도를 포함하여 레이블링을 진행한 실험은 SSD의 크기와 상관없이 Write Amplification의 개선이 이루어졌음을 확인할 수 있고, 빈도를 포함하지 않고 레이블링을 진행한 실험은 1GB의 SSD에서는 오히려 WA가 증가한 모습이며, 1.5GB의 SSD에서는 WA 개선이 이루어졌다. 실험에 사용된 I/O Trace 데이터는 시간이 지남에 따라 빈도의 변화가 크지 않았다고 해석할 수 있다.

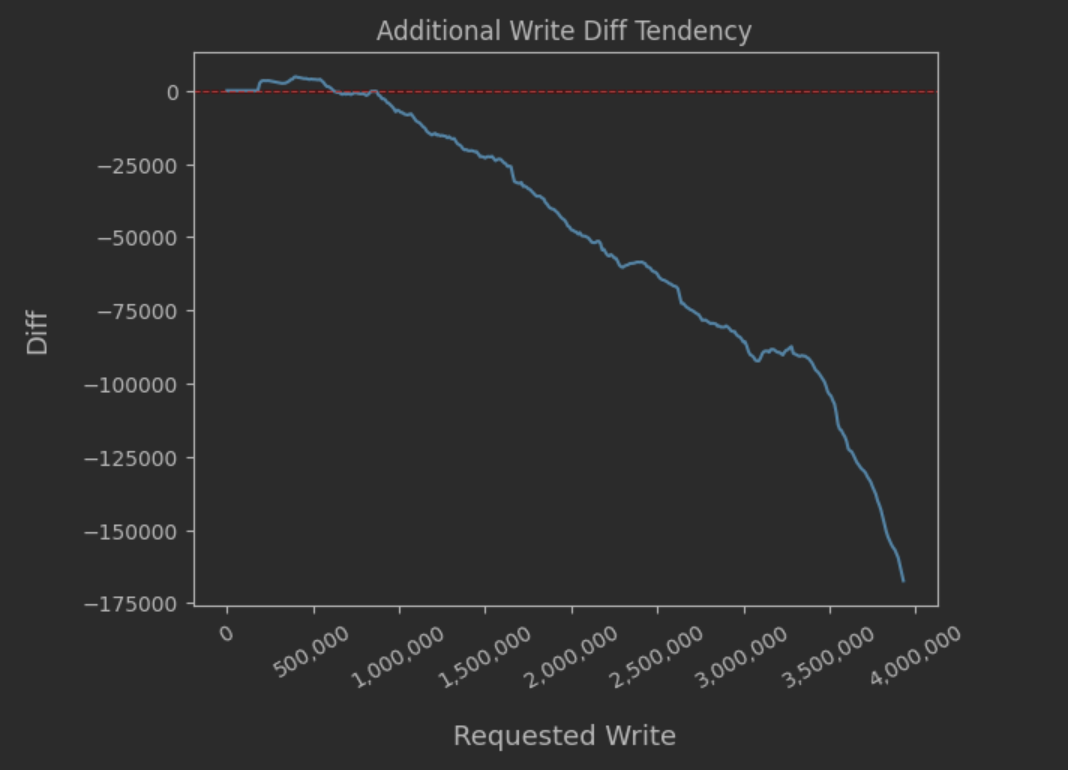


그림 9. Additional Write의 변화 추이

Additional Write Diff는 기계학습을 적용하여 Hot/Warm/Cold Data로 분류했을 때의 Additional Write와, 분류하지 않았을 때의 Additional Write의 차이를 나타낸다. 그림 9는 Requested Write가 증가할수록 Hot/Warm/Cold로 분류하여 시뮬레이션을 진행하는 것이 Additional Write가 적어진다는 것을 나타낸다. 만약 시뮬레이션의 입력 값으로 더 많은 I/O Trace가 있었다면, Hot/Warm/Cold로 분류하여 진행한 시뮬레이션에서 Write Amplification이 더 감소됐을 것으로 예상된다.

## 한계 및 향후 연구 방향

이번 프로젝트에서는 Python으로 구현한 시뮬레이터를 이용하여 실험을 진행했는데, 한 번 시뮬레이션을 진행할 때 속도가 매우 느리다는 단점이 존재했다. 약 394만 건의 데이터를 시뮬레이터에서 동작시켰을 때 약 1시간 50분이 소요됐다. 이에 대한 영향으로 실험에서 많은 파라미터를 적용해보고, 여러 모델을 실험해볼 수 없었다는 한계가 존재했다. 더 나아가, 실험에서 제작했던 모델을 실제 SSD에 이식하여 진행한 프로젝트가 아니라, SSD 시뮬레이터를 작성하여 동작을 모사한 프로젝트였기 때문에 정확성 측면에서 확실하지 않을 수 있다는 한계가 존재했다. 가장 이상적인 기계학습을 탑재한 FTL의 동작으로는 초기에 Block의 Hot/Warm/Cold 비율이 고정된 것이 아닌, 데이터의 패턴에 따라 그 비율이 유동적으로 변화하도록 설계될 것이다. 또한, 실제 FTL에서는 더욱 고도화된 알고리즘으로 Garbage Collection 작업과, 실험에서는 진행하지 않았던 Wear Leveling이 포함되어 있을 것이며, Victim Block을 찾는 전략과 쓰기 시 사용하는 전략이 최적화되어 있을 것이다. 그러므로 실제 SSD에 모델을 이식하여 실험을 진행할 수 있다면, 더욱 프로젝트의 결과가 유의미해질 것이다. 이에 향후 연구 방향으로 실제 SSD에 기계학습을 적용하는 연구를 진행할 수 있을 것이다. 또한, 더욱 더 성능이 좋은 모델을 개발하여 레이블링을 진행한다면, 연구의 성과가 더욱 나아질 것이라고 기대할 수 있다.

# 구성원별 역할 및 개발 일정

## 구성원별 역할

|  |  |
| --- | --- |
| **이름** | **역할** |
| 공희찬 | 학습 데이터 전처리 및 레이블링, 모델 개발,  시뮬레이터 개발 및 테스트, WA 개선도 측정, 포스터 및 보고서 작성 |
| 박영훈 | 데이터 레이블링, 모델 튜닝, 보고서 작성 |
| 박선민 | 모델 튜닝, 포스터 및 보고서 작성 |

## 개발 일정

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5월** | | | | | **6월** | | | | | **7월** | | | | | | **8월** | | | | | **9월** | | | | | **10월** | | |
| **1주** | **2주** | **3주** | **4주** | **5주** | **1주** | **2주** | **3주** | **4주** | **5주** | **1주** | **2주** | **3주** | **4주** | **5주** | **6주** | **1주** | **2주** | **3주** | **4주** | **5주** | **1주** | **2주** | **3주** | **4주** | **5주** | **1주** | **2주** | **3주** |
| **착수보고서** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **데이터 가공 및 모델 공부** | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **1차 데이터 전처리** | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | **레이블링 코드 작성** | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | **2차 데이터 전처리** | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **중간보고서 준비** | | | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **모델 개발, 시뮬레이터 개발** | | | | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **모델 개량** | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **시뮬레이션 진행, 최종 보고서 작성** | | | | | | |

# 참고 문헌

|  |
| --- |
| [1] TDS 데이터복구 [Online]. Available: <https://www.tds119.com/sub0203> (downloaded 2023, Oct. 19) [2] UMASS Trace Repository, OLTP Application I/O [Online]. Available: <https://traces.cs.umass.edu/index.php/Storage/Storage> (downloaded 2023, Sep. 12) |