The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 24, No. 1, pp.31-36, Feb. 29, 2024. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2024.24.1.31
JIIBC 2024-1-5

개인용 모바일 환경의 AI 워크로드 수행을 위한 메모리 참조 분석 및 시스템 설계 방안

Implications for Memory Reference Analysis and System Design to Execute AI Workloads in Personal Mobile Environments

권석민[*], 반효경[**]

Seokmin Kwon[*], Hyokyung Bahn[**]

요 약 최근 AI 기술을 활용하는 모바일 앱이 늘고 있다. 개인용 모바일 환경에서는 메모리 용량의 제약으로 인해 대용량 데이터로 구성된 워크로드의 학습 시 극심한 성능 저하가 발생할 수 있다. 본 논문에서는 이러한 현상을 규명하기 위해 AI 워크로드의 메모리 참조 트레이스를 추출하고 그 특성을 분석하였다. 그 결과 AI 워크로드는 메모리 쓰기 연산 시 약한 시간지역성과 불규칙한 인기편향성 등으로 인해 잦은 스토리지 접근을 발생시켜 모바일 기기의 성능을 저하시킬 수 있음을 확인하였다. 이러한 분석을 토대로 본 논문에서는 AI 워크로드의 메모리 쓰기 연산을 영속 메모리 기반의 스왑 장치를 이용해서 효율적으로 관리할 수 있는 방안에 대해 논의하였다. 시뮬레이션을 통해 본 연구에서 제안한 구조가 기존의 모바일 시스템 대비 80% 이상 입출력 시간을 개선할 수 있음을 보였다.

Abstract Recently, mobile apps that utilize AI technologies are increasing. In the personal mobile environment, performance degradation may occur during the training phase of large AI workload due to limitations in memory capacity. In this paper, we extract memory reference traces of AI workloads and analyze their characteristics. From this analysis, we observe that AI workloads can cause frequent storage access due to weak temporal locality and irregular popularity bias during memory write operations, which can degrade the performance of mobile devices. Based on this observation, we discuss ways to efficiently manage memory write operations of AI workloads using persistent memory-based swap devices. Through simulation experiments, we show that the system architecture proposed in this paper can improve the I/O time of mobile systems by more than 80%.

Key Words : Mobile system, AI workload, memory reference, machine learning

[*]비회원, 이화여자대학교 컴퓨터공학과
[**]정회원, 이화여자대학교 컴퓨터공학과
접수일자 2023년 11월 8일, 수정완료 2024년 1월 8일
게재확정일자 2024년 2월 9일

Received: 8 November, 2023 / Revised: 8 January, 2024 / Accepted: 9 February, 2024
[*]Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

Ⅰ. 서 론

최근 개인용 모바일 환경에서 인공지능 애플리케이션이 실행되는 사례가 늘고 있다[1, 2, 3]. 개인용 모바일 환경은 컴퓨팅 자원뿐 아니라 메모리 크기의 제약으로 인해 대용량 데이터로 구성된 기계학습 워크로드를 수행할 경우 가용 메모리의 부족으로 인한 성능 저하가 발생할 수 있다. 특히, 심층 신경망 워크로드의 학습(training) 단계에서는 작업집합의 크기가 한시적으로 크게 증가하여 메모리 쓰레싱 현상이 발생할 수 있는 것으로 알려져 있다[4]. 또한, 모바일 기기에서 스토리지로 사용되는 NAND 플래시메모리는 많은 쓰기 연산이 발생할 경우 급격한 성능 저하가 나타날 수 있으며 메모리 부족으로 인한 잦은 스토리지 접근은 전체 시스템의 성능을 더욱 악화시키게 된다.

본 논문에서는 모바일 환경에서 기계학습 워크로드 수행시 발생하는 메모리 성능 저하를 해소하기 위해 인공지능 워크로드의 학습 단계에서 발생하는 메모리 참조 트레이스를 추출하고 이를 분석하였다. 그 결과 기존의 워크로드와는 상이한 인공지능 워크로드만의 고유한 메모리 참조 특성들을 발견하였다.

첫째, 읽기가 쓰기에 비해 더 많은 메모리 참조를 유발하지만, 단위 참조 공간당 메모리 접근의 밀도는 쓰기가 더 높은 것을 확인할 수 있었다. 둘째, 쓰기의 인기편향성은 최상위 참조 페이지들과 이를 제외한 페이지들 간에 전혀 다른 양상을 나타내었다. 최상위 페이지들은 쓰기 횟수가 매우 높고 균일한 반면 이를 제외한 페이지들은 인기도의 변동성이 매우 크게 나타났다. 셋째, 읽기의 시간지역성이 일관성 있게 강한 것과 달리 쓰기의 경우 시간지역성이 약하고 특정 순위 이후에는 불규칙한 형태를 나타내었다. 이러한 분석에 따라 인공지능 워크로드는 전통적인 메모리 관리 기법 하에서 쓰기 연산을 효율적으로 관리하기 어려우며, 인기 페이지를 식별하고 이를 유지하기 위해 더 많은 메모리 공간이 필요하다. 특히, 하층부의 스토리지로 플래시메모리가 사용되는 모바일 환경에서는 메모리 공간이 부족할 경우 스토리지에 많은 쓰기 연산을 발생시켜 전체 시스템의 성능을 심각하게 저하시키게 된다.

이러한 분석 결과를 토대로 본 논문에서는 제한된 메모리 자원을 가진 개인용 모바일 환경에서 인공지능 워크로드 수행시 학습 구간에서 발생하는 메모리 쓰레싱을 해소하고 플래시 스토리지에 발생하는 대량의 쓰기 오버헤드를 줄이기 위한 시스템 구조에 대해 살펴 본다. 제안하는 시스템 구조는 소량의 영속 메모리(persistent memory)를 모바일 기기의 쓰기 가속용 스왑장치로 활용하여 인공지능 워크로드의 학습 시 발생하는 메모리 성능 저하를 해소할 수 있음을 보인다. 시뮬레이션 실험을 통해 메모리 풋프린트의 10% 이내의 영속 메모리를 스왑 장치로 추가함으로써 기존 모바일 시스템 대비 80% 이상의 입출력 시간이 감소됨을 확인하였다.

표 1. AI 워크로드 및 메모리 참조 특성
Table 1. AI workloads and memory access characteristics

학습 및 문제 유형		데이터셋		사이즈 (읽기 : 쓰기)	빈도 (읽기 : 쓰기)
지도	회귀	Boston House Price (BHP)	Pytorch	4.66 : 1	2.54 : 1
			Scikit-learn	3.86 : 1	1.40 : 1
비지도	클러스터링	Iris Segmentation (IS)	Pytorch	6.26 : 1	3.18 : 1
			Scikit-learn	3.62 : 1	1.99 : 1

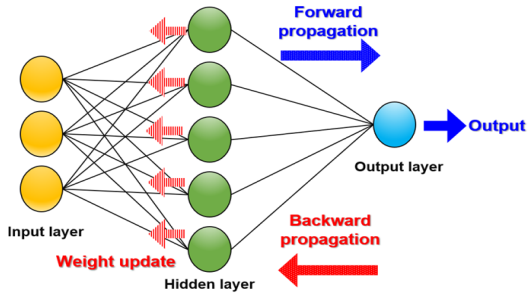
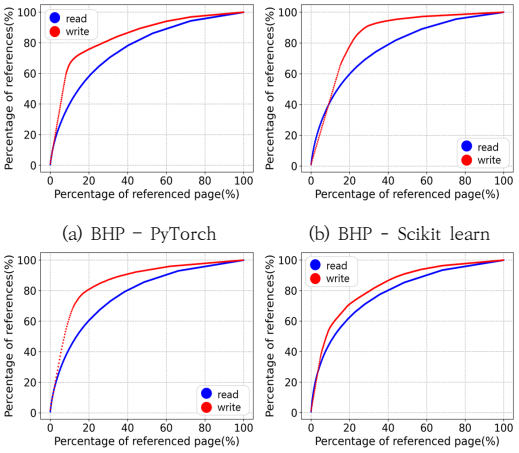


그림 1. 인공지능 워크로드에서의 학습 과정
Fig. 1. Training process in AI workloads

Ⅱ. AI 워크로드의 메모리 참조 분석

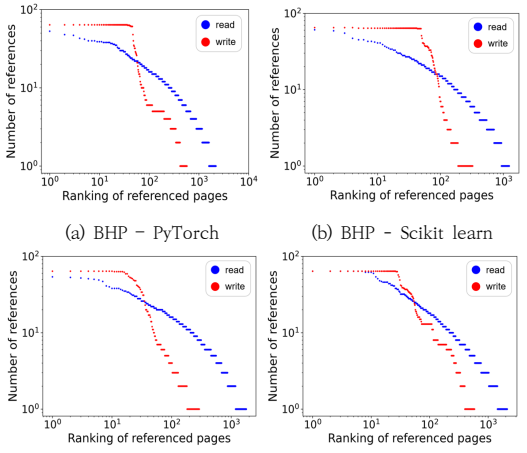
본 논문에서는 기계학습의 대표적인 2가지 유형인 지도학습과 비지도학습 워크로드에서 학습(training) 구간의 메모리 참조 트레이스를 추출하고 이를 분석하였다. 데이터셋으로는 지도학습의 회귀 문제로 잘 알려진 Boston House Price Dataset[5]과 비지도학습의 클러스터링 문제로 잘 알려진 Iris Segmentation Dataset[6]을 사용하였다. 실제 워크로드는 대표적인 오픈소스 기반 머신러닝 라이브러리인 Scikit-learn[7]과 PyTorch[8]로 구현된 모델로 수행하였으며, 텍스트 및 실수로 구성된 데이터셋을 포함한다. 메모리 참조 트레이스는 메모리 분석 도구인 Valgrind의 Cachegrind 모듈을 수정하여 추출하였다[9].



(a) BHP - PyTorch (b) BHP - Scikit learn

(c) IS - PyTorch (d) IS - Scikit learn

그림 2. 페이지 참조의 상위 백분율에 따른 누적 참조분포
Fig. 2. Cumulative reference distribution of top percent of pages referenced



(a) BHP - PyTorch (b) BHP - Scikit learn

(c) IS - PyTorch (d) IS - Scikit learn

그림 3. 페이지의 인기도 순위에 따른 참조 횟수
Fig. 3. Number of references versus the page's ranking

표 1은 상기의 워크로드에 대한 학습 과정에서 발생한 메모리 참조 트레이스를 추출한 대략적인 결과를 보여주고 있다. 표에서 보는 것처럼 모든 경우에 있어 읽기가 쓰기에 비해 높은 비율을 나타내었으며, 메모리 풋프린트의 경우 평균 3.6배, 메모리 접근 횟수의 경우 평균 2.1배로 나타났다. 이는 쓰기가 읽기에 비해 상대적으로 더 적은 메모리 영역에 대해 발생하지만 단위 공간 당 접근 빈도는 더 높다는 것을 의미한다. 이러한 메모리 접근 경향은 그림 1에서 보는 것처럼 읽기의 경우 학습의 주기(epoch)마다 다수의 은닉층(hidden layer)을 거치면서 순전파 과정에서 예측값을 계산하고 역전파 과정에서 가중치값을 갱신하는 과정에 많은 양의 페이지에 대한 읽기가 필요하며, 이에 비해 쓰기는 상대적으로 적은 페이지들에 빈번하게 발생하기 때문으로 추측할 수 있다.

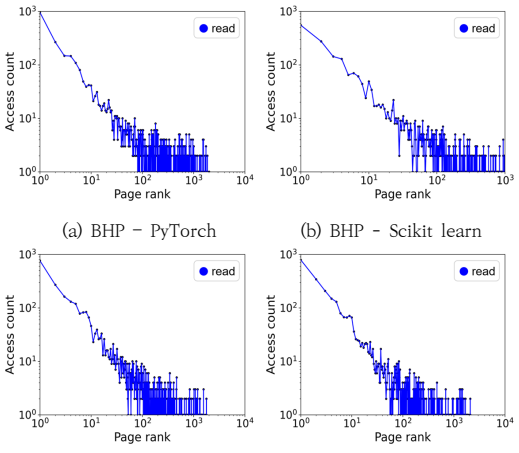
그림 2는 각 메모리 페이지를 참조 횟수에 따라 정렬했을 때 상위 몇 퍼센트의 페이지가 전체 누적 참조의 몇 퍼센트를 나타내는지를 보여주고 있다. 그래프에서 보는 것처럼 읽기의 경우 상위 20%의 페이지가 전체 참조의 60% 정도를 나타내며, 곡선의 형태가 완만하게 증가하는 것을 확인할 수 있다. 이해 비해 쓰기의 경우 곡선에 변곡점이 존재하며, 상위 페이지들에 대한 편향도가 매우 심한 것을 확인할 수 있다. 구체적으로는 상위 20%의 페이지가 전체 참조의 80% 정도를 차지하였다.

그림 3은 메모리 페이지의 인기도 순위에 따른 참조횟수를 보여주고 있다. 그림에서 보는 것처럼 읽기의 경우 인기도 순위 증가에 따른 접근 횟수 감소가 일정한 경향성을 나타내었으며 이는 인기 편향성을 모델링하는 전형적인 형태인 Zipf 분포와 유사한 결과를 나타내었다. 이에 비해 쓰기의 경우 최상위 순위에서는 일정 순위까지 참조 횟수가 거의 동일하게 나타났으며, 이후부터 매우 가파르게 참조횟수가 낮아지는 다소 불규칙한 형태를 나타내었다.

효율적인 메모리 관리를 위해서는 워크로드의 시간지역성 분석을 통한 재참조 가능성의 예측이 중요하다. 그림 4와 5는 메모리 페이지의 시간지역성 순위에 따른 재참조 횟수를 읽기 및 쓰기 연산에 대해 각각 조사한 그래프이다. 그림에서 x축은 페이지의 시간지역성 순위(즉, 가장 최근에 참조된 페이지가 1위)를 나타내며, y축은 해당 x축 순위에서 재참조가 발생한 횟수를 나타낸다. 통상적으로 최근에 참조된 페이지일수록 다시 참조되는 경향이 높으므로 그래프는 감소함수 형태를 나타낸다. 그림 4에서 보는 것처럼 읽기의 경우 뚜렷한 감소함수 형태로 시간지역성이 강한 것을 확인할 수 있으나, 쓰기의 경우 그림 5에 보는 것처럼 감소 후 다시 증가하는 등 다소 불규칙한 경향을 확인할 수 있다. 이는 시간지역성을 활용하는 일반적인 메모리 관리 방식 하에서 쓰기 성능 개선을 위해서는 더 큰 용량의 메모리가 필요함을 시사한다.

III. 시스템 구조

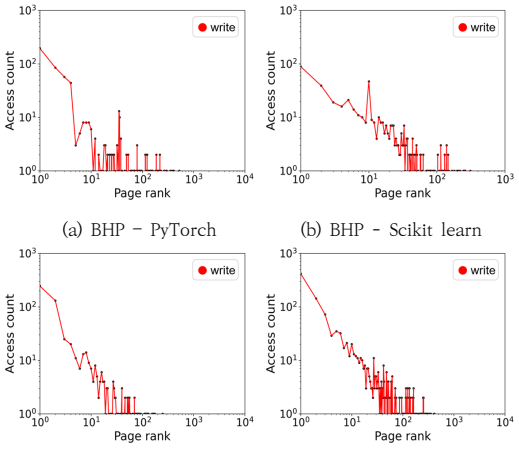
본 장에서는 II장에서 분석한 인공지능 워크로드의 메모리 참조 특성을 반영할 수 있는 개인형 모바일 환경의



(a) BHP - PyTorch (b) BHP - Scikit learn

(c) IS - PyTorch (d) IS - Scikit learn

그림 4. 메모리 읽기 연산의 시간지역성 분석
Fig. 4. Analysis of temporal locality on memory reads

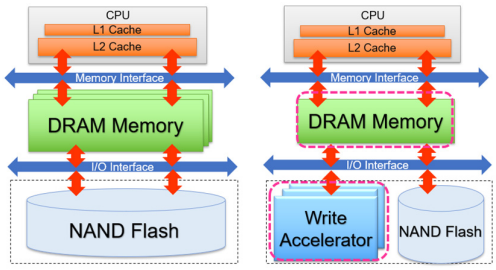


(a) BHP - PyTorch (b) BHP - Scikit learn

(c) IS - PyTorch (d) IS - Scikit learn

그림 5. 메모리 쓰기 연산의 시간지역성 분석
Fig. 5. Analysis of temporal locality on memory writes

시스템 구조에 대해 살펴본다. 제한된 메모리 크기를 갖는 개인형 모바일 환경에서는 시간지역성이 약하고 인기 편향성의 변화가 큰 쓰기 연산을 효율적으로 관리하는 것이 쉽지 않다. 특히, 가상메모리 시스템의 관리는 버퍼 캐쉬나 스토리지 캐쉬처럼 페이지의 정확한 참조 기록을 활용하는 것이 불가능하고 1비트의 최근 참조 여부만을 활용하는 것이 일반적이므로 쓰기 연산의 특성을 활용하는 정교한 관리가 어렵다[10]. 한편, 쓰기 연산이 메모리를 통해 효율적으로 흡수되지 않을 경우 플래시 스토리지로 많은 양의 입출력 연산이 전파되어 모바일 시스템의 성능은 극심하게 저하될 수 있다. 이를 해결하기 위해서는 쓰기 연산을 유발하는 페이지들을 유지할 더 많은 메모리 공간이 필요하나, D램의 집적도 한계와 전력 소모 문제로 모바일 시스템의 메모리 공간을 늘리는 것 또한 현실적이지 않다.



(a) 기존 모바일 구조 (b) 제안하는 시스템 구조

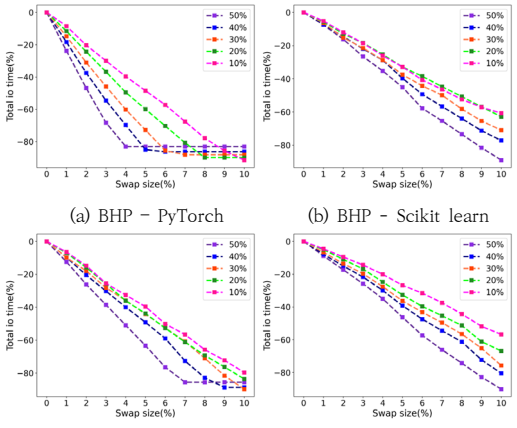
그림 6. 모바일 시스템의 아키텍처 비교
Fig. 6. Comparison of mobile system architectures

따라서, 본 논문에서는 기존의 모바일 시스템이 채택하는 NAND 플래시에 소량의 영속메모리 기반의 스왑 장치를 추가할 경우 인공지능 워크로드의 메모리 쓰기 연산을 어떻게 효율적으로 완충할 수 있는지에 대해 살펴본다. 그림 6은 기존의 개인용 모바일 시스템 구조와 영속 메모리에 기반한 쓰기 가속용 스왑 장치를 추가한 시스템 구조를 비교해서 보여주고 있다. 최근 모바일 시스템을 위한 영속 메모리인 eMRAM이 삼성전자 등을 통해 연구 개발이 이루어지고 있으며[11], 본 논문에서는 이러한 영속 메모리가 추가된 모바일 시스템에서 인공지능 워크로드의 수행 시 메모리 성능이 어떻게 변화하는지 시뮬레이션을 통해 정량적으로 분석하고자 한다.

IV. 성능 평가

본 장에서는 시뮬레이션을 통해 III장에서 제안한 시스템 구조에 대한 성능을 평가한다. 본 논문의 실험에서는 표 1에서 제시한 4종의 인공지능 워크로드에 대한 메모리 참조 트레이스를 재현하였으며, 전체 시스템에서 각 워크로드에 할당되는 메모리 용량의 효과를 확인하기 위해 워크로드별 최대 메모리 풋프린트의 10%에서 50%까지 메모리 크기를 변화시키고, 쓰기 가속용 스왑 장치의 경우 풋프린트의 1%에서 10%까지 변화시키며 실험을 진행하였다. 스왑 장치로는 삼성전자의 eMRAM을 가정하였다[11].

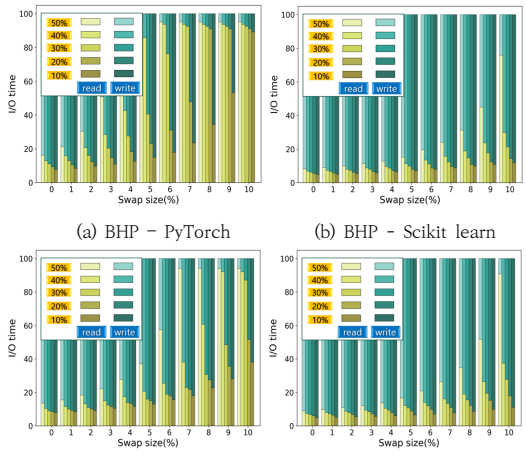
그림 7은 각 메모리 용량 하에서 스왑 장치의 크기가 변함에 따른 전체 입출력 시간을 보여준다. 그래프에서



(a) BHP - PyTorch (b) BHP - Scikit learn

(c) IS - PyTorch (d) IS - Scikit learn

그림 7. DRAM과 스왑 장치 크기에 따른 입출력 시간
Fig. 7. I/O time as a function of DRAM and swap device sizes



(a) BHP - PyTorch (b) BHP - Scikit learn

(c) IS - PyTorch (d) IS - Scikit learn

그림 8. DRAM과 스왑 장치 크기에 따른 읽기 쓰기 시간
Fig. 8. Read/write latency as a function of DRAM and swap sizes

*x*축은 스왑 장치의 크기를 나타내며 *y*축은 스왑 장치를 사용하지 않는 시스템 대비 입출력 시간의 감소율을 나타낸다. *x*축의 값이 0인 지점은 스왑 장치를 사용하지 않는 기존 모바일 시스템을 의미한다. 그림에서 보는 것처럼 4종의 워크로드 모두 메모리 풋프린트의 10% 이내 크기의 스왑 장치를 추가함으로써 기존 모바일 시스템 대비 80% 이상의 입출력 시간이 감소됨을 확인할 수 있다. PyTorch 기반의 인공지능 워크로드에서는 풋프린트의 5% 내외로 스왑 장치를 사용하여 성능 개선이 가능한 최대폭만큼 입출력 시간을 줄일 수 있었으며, Scikit-learn 기반의 워크로드에서는 1%부터 10%까지 스왑 장치의 크기를 증가시킬수록 입출력 시간의 감소 효과가 꾸준히 나타나는 것을 확인할 수 있었다.

그림 8은 입출력 시간 중 읽기 연산과 쓰기 연산이 차지하는 비율을 분리해서 보여주고 있다. 그림에서 보는 것처럼 고속 스왑 장치의 크기를 증가시킴에 따라 전체 입출력 시간에서 쓰기 연산이 차지하는 비율이 점점 줄어드는 것을 확인할 수 있다. 이를 그림 7의 결과와 종합해보면 입출력 시간에서 쓰기 연산이 차지하는 비율이 줄어들수록 전체 입출력 시간이 크게 감소한다는 사실을 알 수 있다. 그 이유는 대량의 쓰기 연산이 플래시메모리의 성능을 저하시키고 이를 고속 스왑 장치가 흡수하기 때문으로 볼 수 있다. 또한, PyTorch 기반의 AI 워크로드가 Scikit-learn 기반 AI 워크로드에 비해 입출력 시간에서 쓰기 연산의 비율이 더 크게 감소하는 것을 확인할 수 있었다.

V. 결 론

본 논문에서는 인공지능 워크로드의 학습 구간에서 발생하는 메모리 참조의 특성을 분석하고, 이를 기반으로 개인용 모바일 환경에서 효율적으로 학습이 이루어질 수 있는 시스템 구조에 대해 살펴보았다. 인공지능 워크로드는 메모리 쓰기 연산 시 약한 시간지역성과 불규칙한 인기편향성으로 인해 메모리 크기가 작은 모바일 환경에서 NAND 플래시 스토리지로 많은 입출력을 발생시켜 성능 저하를 초래할 수 있음을 확인하였다. 이에 본 논문에서는 영속 메모리를 사용한 고속 스왑 장치를 통해 모바일 시스템의 메모리 쓰기 연산을 얼마나 효율적으로 관리할 수 있는지를 확인하였다. 4종의 인공지능 워크로드를 이용한 시뮬레이션 실험을 통해 본 연구에서 제안한 시스템 구조가 기존의 모바일 시스템 대비 80% 이상 입출력 시간을 개선할 수 있음을 보였다. 본 연구의 메모리 참조 분석 및 시스템 구조가 향후 인공지능 워크로드를 위한 개인용 모바일 환경 구축에 도움이 될 수 있기를 기대한다.

References

[1] K. Lee, H. Jung, and S. Lee, “Anomaly detection method of user trajectories based on deep learning technologies,” JKIIT, vol. 20, no. 11, pp. 101-116, 2022. DOI: https://doi.org/10.14801/jkiit.2022.20.11.101

[2] S. Ki, G. Byun, K. Cho, and H. Bahn, "Co-optimizing CPU voltage, memory placement, and task offloading for energy-efficient mobile systems," IEEE Internet of Things Journal, vol. 10, no. 10, pp. 9177-9192, 2023. DOI: https://doi.org/10.1109/JIOT.2022.3233830

[3] S. Kim, W. Hur, and J. Ahn, "A progressive web application for mobile crop disease diagnostics based on transfer learning," Journal of the Korea Academia-Industrial cooperation Society (JKAIS), vol. 23, no. 2 pp. 22-29, 2022.
DOI: https://doi.org/10.5762/KAIS.2022.23.2.22

[4] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. Keckler, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," IEEE/ACM MICRO Conf., pp. 1-13, 2016.
DOI: https://doi.org/10.1109/MICRO.2016.7783721

[5] S. Sanyal, S. Kumar Biswas, D. Das, M. Chakraborty, and B. Purkayastha, "Boston house price prediction using regression models," IEEE CONIT Conf., pp. 1-6, 2022.
DOI: https://doi.org/10.1109/CONIT55038.2022.9848309

[6] W. Zhang, X. Lu, Y. Gu, Y. Liu, X. Meng, and J. Li, "A robust iris segmentation scheme based on improved U-Net," IEEE Access, vol. 7, pp 85082-85089, 2019.
DOI: https://doi.org/10.1109/ACCESS.2019.2924464

[7] Scikit-learn Machine Learning in Python, https://scikit-learn.org/stable/

[8] PyTorch, https://pytorch.org/

[9] Cachegrind: A high-precision tracing profiler, https://valgrind.org/docs/manual/cg-manual.html

[10] H. Bahn, "Design of a memory management policy separating the characteristics of read and write references," The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), vol. 23, no. 1, pp. 71-76, 2023.
DOI: https://doi.org/10.7236/JIIBC.2023.23.1.71

[11] K. Suh, J. Lee, H. Shin, J. Lee et al., "12.5 Mb/mm2 embedded MRAM for high density non-volatile RAM applications," IEEE Symp. VLSI Technology, 2021.

저 자 소 개

권 석 민(비회원)

- 2018년 2월 : 이화여자대학교 컴퓨터공학과 학사
- 2021년 2월 : 서울대학교 컴퓨터공학부 석사
- 2022년 3월 ~ : 이화여자대학교 컴퓨터공학과 박사과정
- 주관심분야 : 운영체제, 인공지능시스템, 스토리지 시스템

반 효 경(정회원)

- 1997년 2월 : 서울대학교 계산통계학과 학사
- 1999년 2월 : 서울대학교 전산과학과 석사
- 2002년 2월 : 서울대학교 컴퓨터공학부 박사.
- 2002년 9월 ~ : 이화여자대학교 컴퓨터공학과 교수.
- 주관심분야 : 운영체제, 스토리지시스템, 임베디드시스템

※ This work was partly supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2021-0-02068, Artificial Intelligence Innovation Hub) and (No.RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha University)).