CXL 메모리를 위한 딥러닝 기반 프리페치 기법 연구

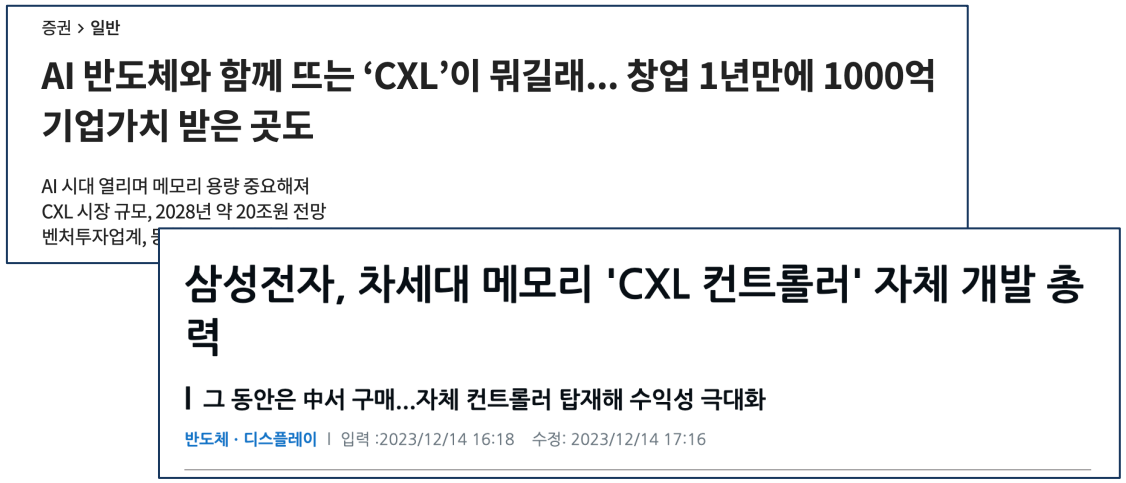
2024/05/27

숭실대학교 AI융합학부
김성연

차례

프로젝트 배경 및 목적

- **CXL(Compute Express Link) : 메모리 용량 한계 극복 위한 차세대 인터페이스**
  - ✓ 여러 개의 인터페이스를 하나로 통합 → 확장성 및 데이터 처리 속도 확보
- **CXL 메모리**
  - ✓ CXL을 적용한 메모리 기술 기대
  - ✓ 플래시 메모리의 메모리 접근 시간으로 인한 성능 저하 → 공격적 프리페치 필요



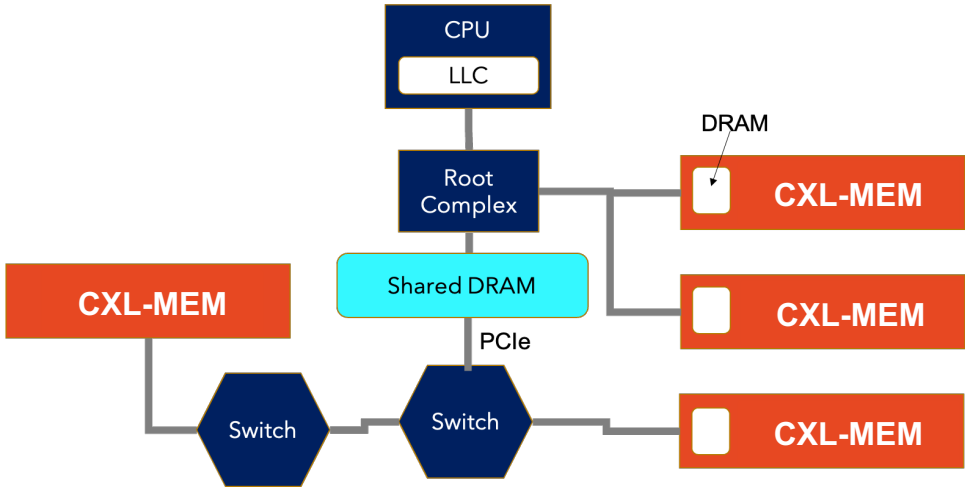


그림1. CXL 메모리 적용 Architecture

프로젝트 배경 및 목적

- 기존 프리페치 기법의 한계
  - ✓ 기존 프리페치 기법 : Locality에 초점을 맞추어 메모리 접근 패턴을 찾아내고자 함
  - ✓ Locality : 물리메모리 주소 << 가상메모리 주소
  - ✓ CXL 메모리 : 물리 메모리 주소 요청
  - ✓ Locality에 기반한 기존 기법 사용 시, 성능 저하 문제 발생

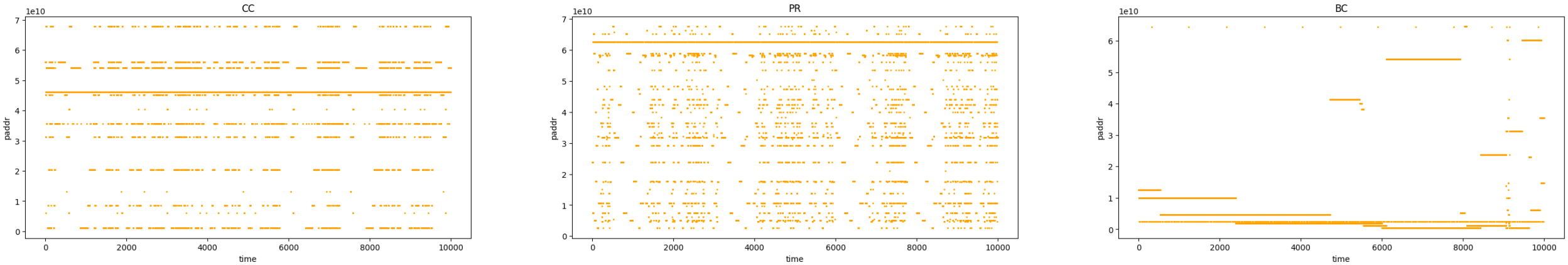


그림2. Trace에 따른 메모리 접근 분포

프로젝트 배경 및 목적

- **기존 ML 기반 프리페치 기법과의 차이**
  - ✓ **CXL 메모리 프리페치 : 물리 메모리 주소 사용 ⇔ 기존 ML 기반 프리페치 : 가상 메모리 주소 사용**
  - ✓ **CXL 메모리 프리페치 : 호스트 정보를 사용 불가 ⇔ 기존 ML 기반 프리페치 : 호스트 정보 사용**

Prefetcher	Publication	Addr	Host Info.	Target Layer	Impl.
Context-based Prefetching with RL	ISCA'15	Virtual	PC	L2 form LLC	HW
Delta-LSTM	ICML'18	Virtual	PC	L2 from LLC	HW
Voyager	Asplos'21	Virtual	PC	On-chip cache form DRAM	HW

Context-based Prefetching with RL[1] Delta-LSTM[2] Voyager[3]

표1. ML 기반 프리페치 기법 특성 요약

프로젝트 배경 및 목적

(Why)

플래시 메모리의 메모리 접근 시간으로 인한
CXL 메모리의 성능 저하를 개선하기 위해

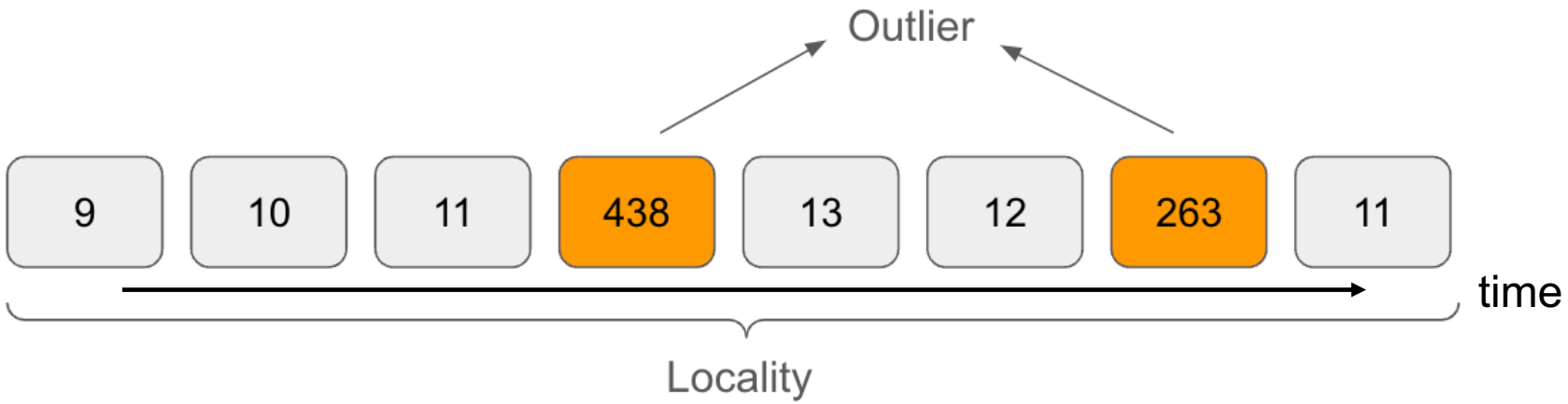
+

(What)

Locality에 벗어난 Outlier를 예측할 수 있는 프리페치 기법을 개발하자

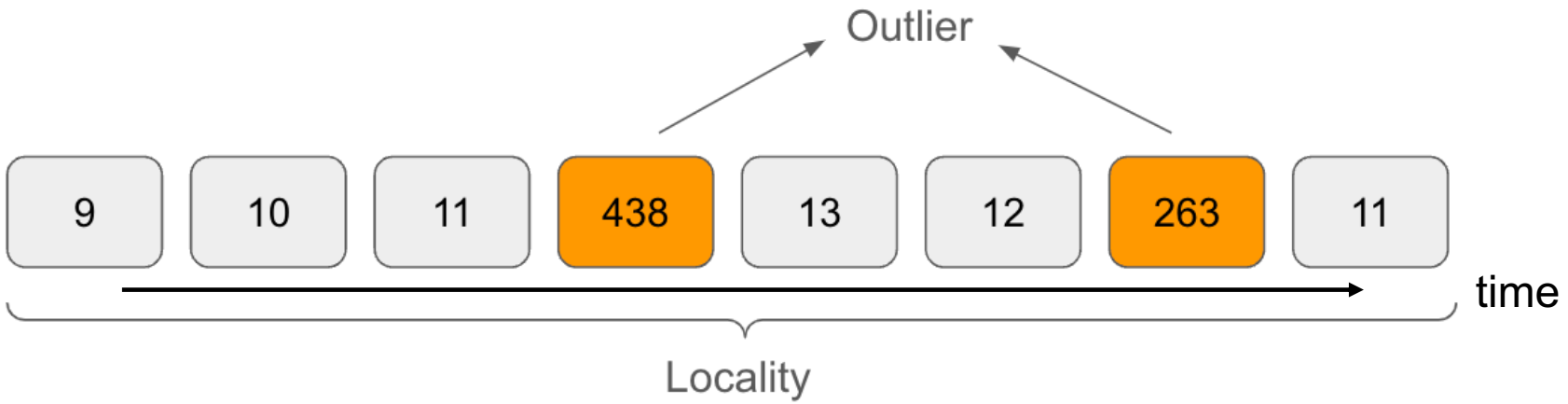
아이디어 제시

- 기존 프리페치 + 딥러닝 모델을 활용한 프리페치
  - ✓ 기존 프리페치의 miss 영역을 커버하기 위해 딥러닝 모델 사용
  - ✓ Conventional : locality capture
  - ✓ DL Model : outlier capture



아이디어 구체화

- **Leap[4] + Attention-based LSTM을 활용한 프리페치**
  - ✓ **Leap의 miss 영역을 커버하기 위해 Attention-based LSTM 사용**
  - ✓ **Leap : locality capture**
  - ✓ **Attention-based LSTM Model : outlier capture**



※ Leap : 최근에 발생한 demand miss의 히스토리에 따라 offset과 aggressiveness를 동적으로 바꾸는 방식

아이디어 구체화

- **Leap 프리페치**
  - ✓ 대표 비교군을 설정하고자 함
  - ✓ 기존 프리페치 중 가장 우수한 성능을 보임
  - ✓ CXL 메모리에 적용한 결과, Outlier 예측에 대한 추가 개선이 필요한 것으로 판단

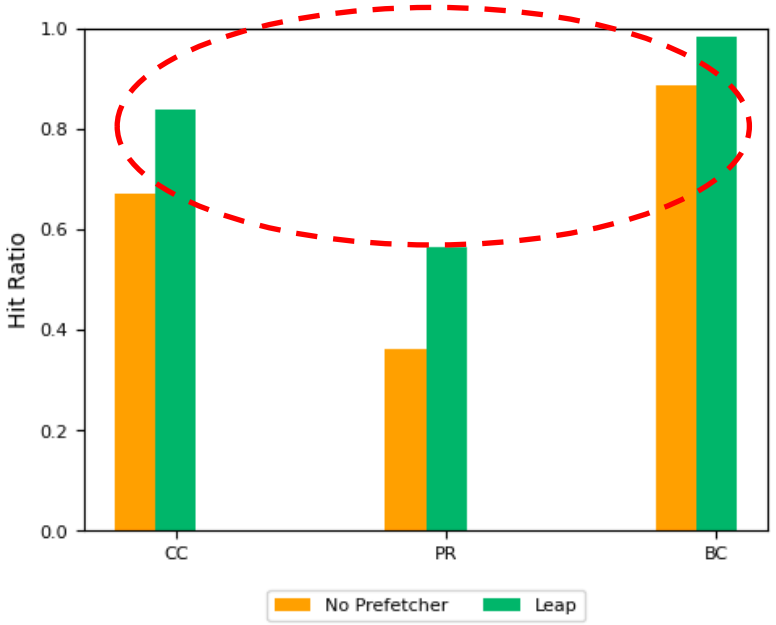


그림3. Trace에 따른 Leap 프리페치 Hit Ratio

아이디어 구체화

- **Attention-based LSTM**
  - ✓ **LSTM**
    - **NLP(Natural Language Processing) 분야에서 주로 사용되는 시계열 학습 기법**
  - ✓ **Attention Mechanism**
    - **앞선 Context 중 특정 부분에 가중치를 주어 다음 예측을 할 수 있도록 함**

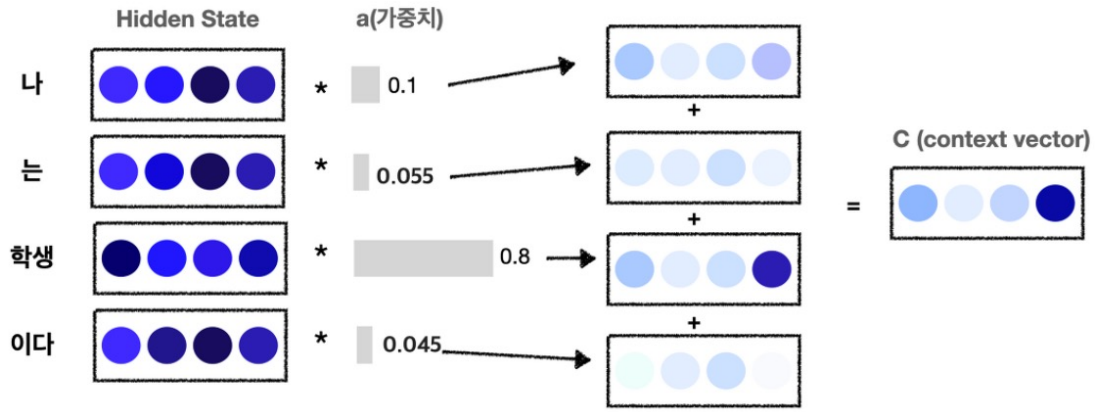


그림4. Attention 기법 적용 예시

➔ **앞선 메모리 주소 요청 정보들을 사용하여 다음에 요청될 주소를 예측할 때,**

흐름 상 영향력이 있을 것 같은 주소들에 더 가중치를 부여하기 위함

트레이스 재구성

- GAP(Graph Algorithms and Applications) Traces
  - ✓ Modified ChampSim for ML Prefetching Competition에서 제공
  - ✓ 가상 주소 → 물리 주소
    - Page 단위로 나누어 랜덤 매핑 수행
    - Huge Page : Locality 특성 일부 보존 가능 → 임베딩 오버헤드를 줄일 수 있음

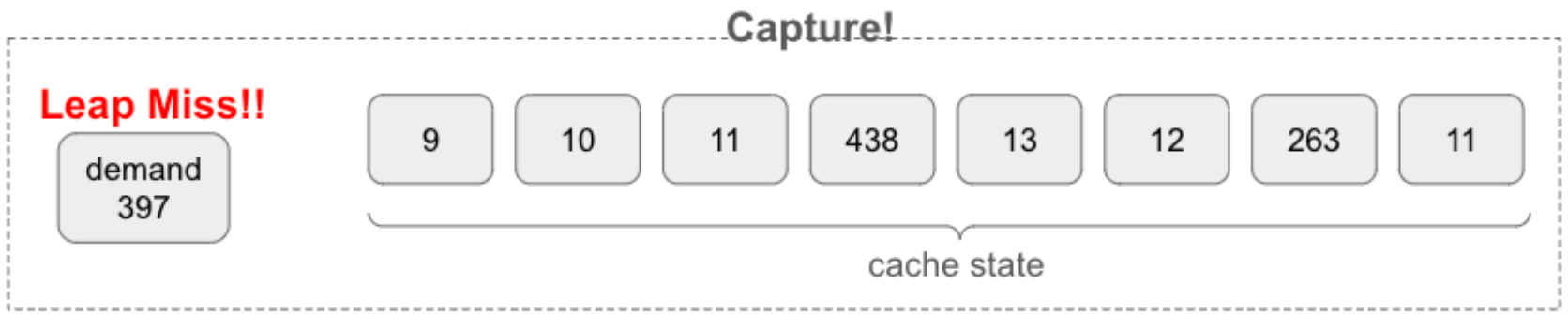
```
va,pa
44977833418496,584581451915008
44977833422656,584581451919168
44977833429056,584581451925568
44977833428800,584581451925312
44977833428928,584581451925440
44977833428992,584581451925504
44977833433280,584581451929792
44977833437376,584581451933888
44977833422400,584581451918912
44977833418304,584581451914816
44977833447488,584581451944000
```

그림5. CC 트레이스의 가상 주소와 물리 주소 사이 매핑 테이블 예시

데이터셋 구성 및 전처리

- **Cache State를 통한 학습 데이터셋 구성**
  - ✓ **Input Data : Leap의 Demand Miss 발생 시, Cache State**
  - ✓ **Output Data : 다음에 발생할 Leap의 Demand Miss 주소**

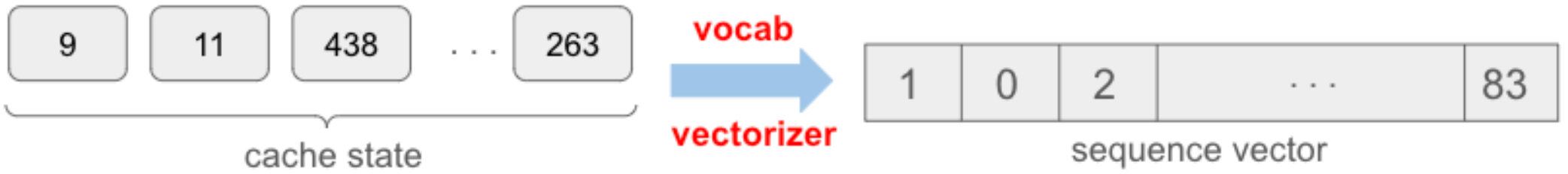
- **Cache State 사용 이유**
  - ✓ **Leap이 놓친 Outlier에 대한 예측이 목표**
  - ✓ **Cache State : Leap의 Demand Miss 발생 시점 부근 요청 주소 정보 담고 있음**



데이터셋 구성 및 전처리

- **Vocabulary + Vectorizer를 통한 전처리**
  - ✓ **Vocabulary : 물리 주소를 정수 인덱스화**
    - **ex) token_to_idx {'11' : 0, '9' : 1, '438' : 2, … , '263' : 83, … , *'paddr' : N*}**
  - ✓ **Vectorizer : Cache State (128개의 주소) 를 벡터화**

⇒ Vocabulary + Vectorizer : Cache State를 주소의 집합이 아닌 벡터화 된 하나의 문장으로 나타냄



시스템 구조 설계

- **Cache state-based LSTM (CLSTM)**
  - ✓ 페이지 단위 (16KB) 접근 → 공격적 프리페치
  - ✓ Leap 프리페치와 함께 동작 → Leap의 결과와 겹치는 데이터는 프리페치X

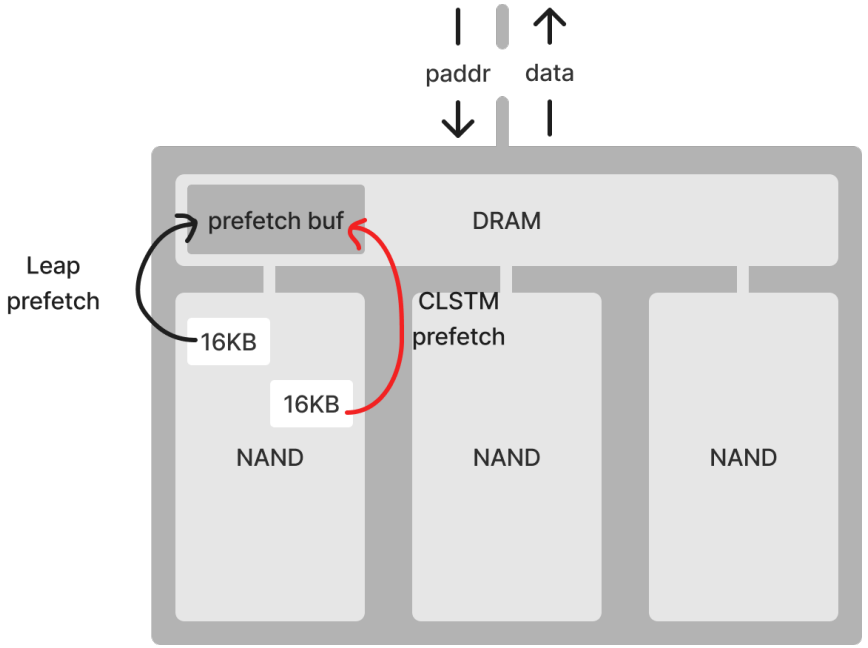


그림6. CLSTM 프리페치 적용 시, 시스템 구조 도식화

시스템 구조 설계

- **Cache state-based LSTM (CLSTM)**
  - ✓ **Vocabulary + Vectorizer : 주소의 벡터화**
  - ✓ **Embedding Layer : 주소 간의 관계 파악**
  - ✓ **Attention-based LSTM Layer : 주소 접근 패턴 파악 및 다음 주소 예측**
  - ✓ **가장 확률이 높은 Top K 개의 데이터 프리페치 수행**

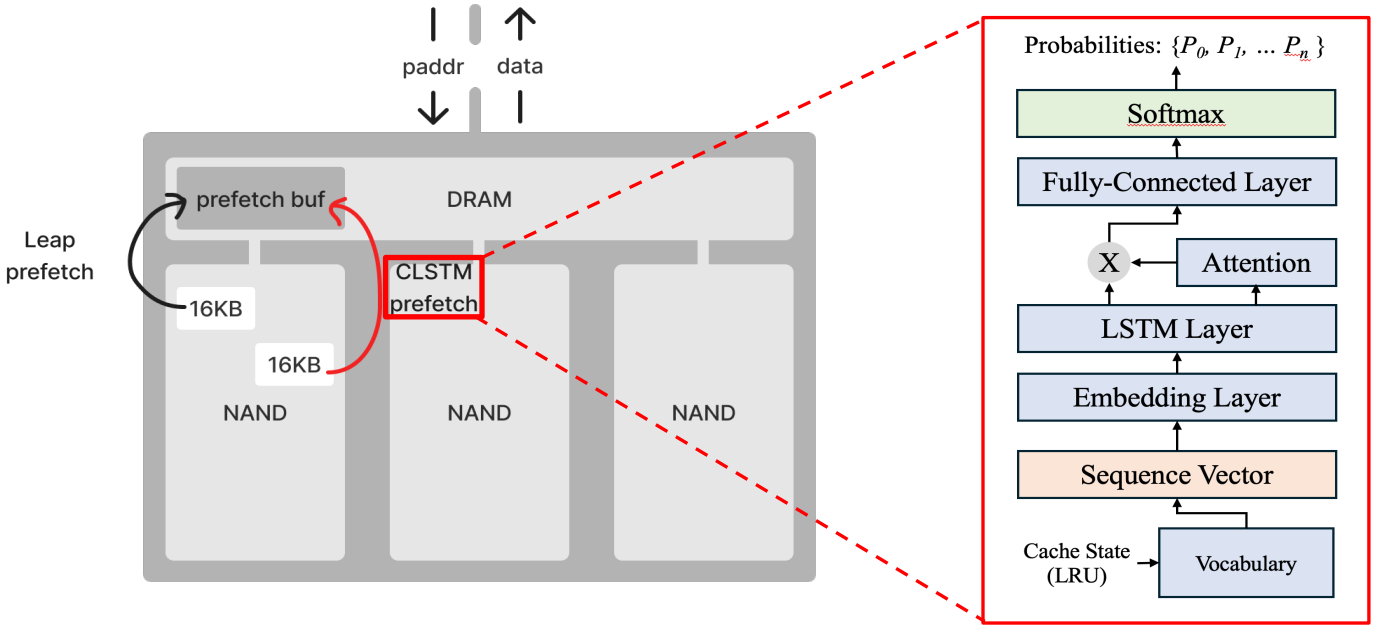


그림7. CLSTM 프리페치 적용 시, 시스템 상세 구조 도식화

시스템 동작 설계

- **CLSTM Top-K 설정**
  - ✓ K가 작으면, Hit Ratio가 비교적 낮아짐
  - ✓ K가 크면, Hit Ratio 높아지지만 Cache Pollution이 심해짐
  - ✓ 사용되는 주소만을 가져오는 것이 중요 → 약 45~50% 커버 가능한 K=10 설정

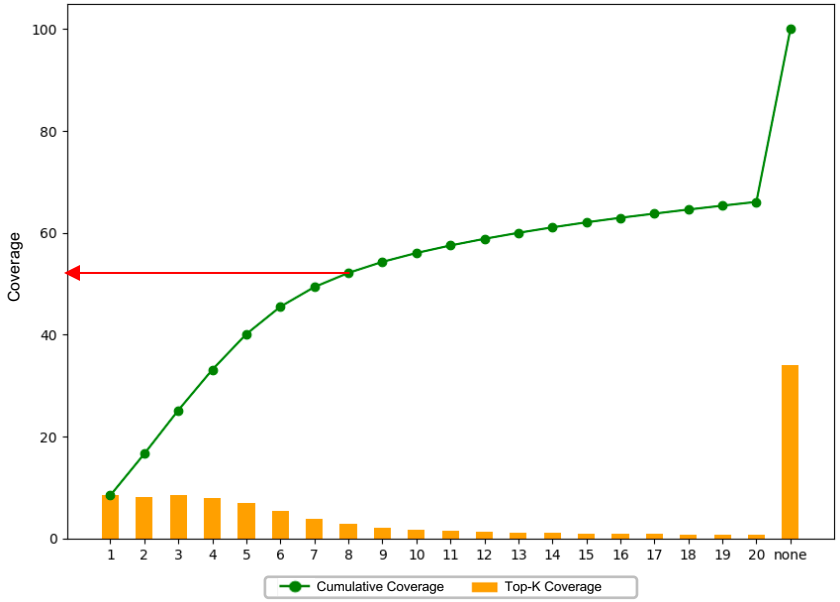


그림8. Top-K에 따른 CC의 CLSTM 프리페치 Hit Ratio

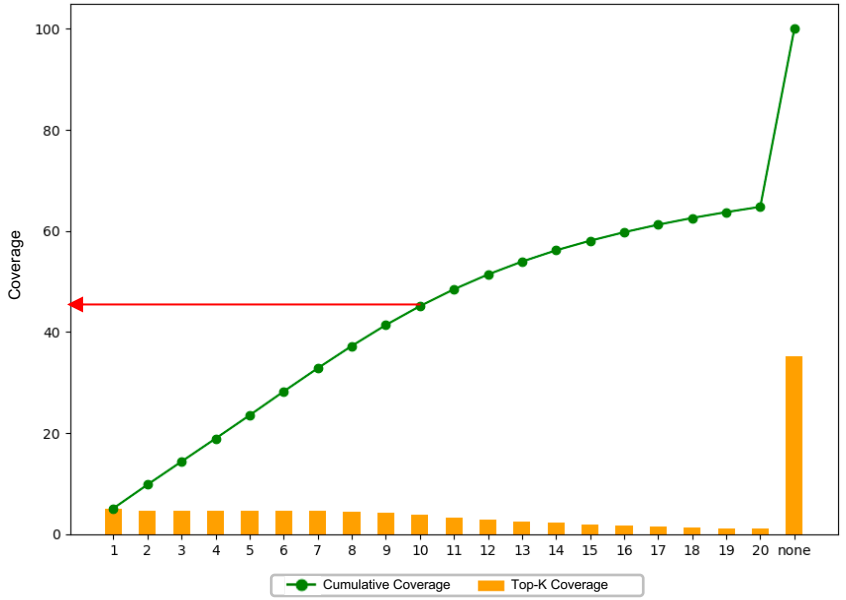


그림9. Top-K에 따른 PR의 CLSTM 프리페치 Hit Ratio

시스템 동작 설계

- CLSTM 동작 유무 설정
  - ✓ CLSTM
    - 매 시점 동작
  - ✓ CLSTM90
    - Hit Ratio 90% 이하인 시점에서만 동작
    - 기존 프리페치의 성능이 뛰어나 CLSTM의 사용이 자원 낭비가 되는 것을 막기 위함

```
if (self.hits + self.pf_hits)/self.refs < 0.9:
    for clstm_pd in clstm_result:
        # if pd not in self.pf_buff:
        pd = int(clstm_pd)
        if pd not in self.pf_buff and pd not in self.dlist:
            self.pf_buff.append(pd)
            pf_data_num += 1
        if len(self.pf_buff) > self.pf_buff_slots:
            self.pf_buff.pop(0) # FIFO
```

그림10. CLSTM90 프리페치 적용 시, 시뮬레이터 코드 발췌

실험 결과 분석

- **실험 환경**
  - ✓ **운영체제 : Ubuntu 20.04**
  - ✓ **시뮬레이터 : Python3.9**
  - ✓ **대표 비교군 : No Prefetcher/ Leap / CLSTM / CLSTM90**
  - ✓ **트레이스 : GAP의 CC(Connected Components) / PR(PageRank) / BC(Betweenness Centrality)**

실험 결과 분석

- **Hit Ratio**
  - ✓ **CC, PR : Leap 대비 약 10%의 Hit Ratio 증가**
  - ✓ **BC : Leap 대비 증감이 크게 나타나지 않음**
    - **기존 Leap 프리페치 성능이 우수함 → Demand miss 데이터셋이 모이지 X**
    - **오히려 프리페치 시, 자원 낭비 발생 → Cache Pollution, Bandwidth 낭비**

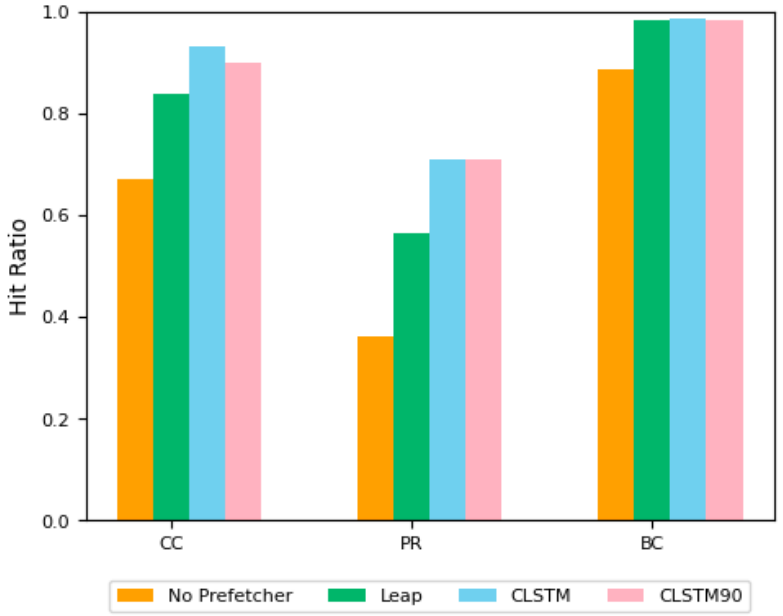


그림11. Trace에 따른 프리페치 Hit Ratio

실험 결과 분석

- **Average Number of Prefetch**
  - ✓ 한 시점에 평균적으로 가지고 오게 되는 페이지 수
- **Cache Pollution**
  - ✓ 사용되지 않고 쫓겨난 페이지 수 / 전체 프리페치 된 페이지 수

➔ CLSTM90을 통해 자원 낭비 방지

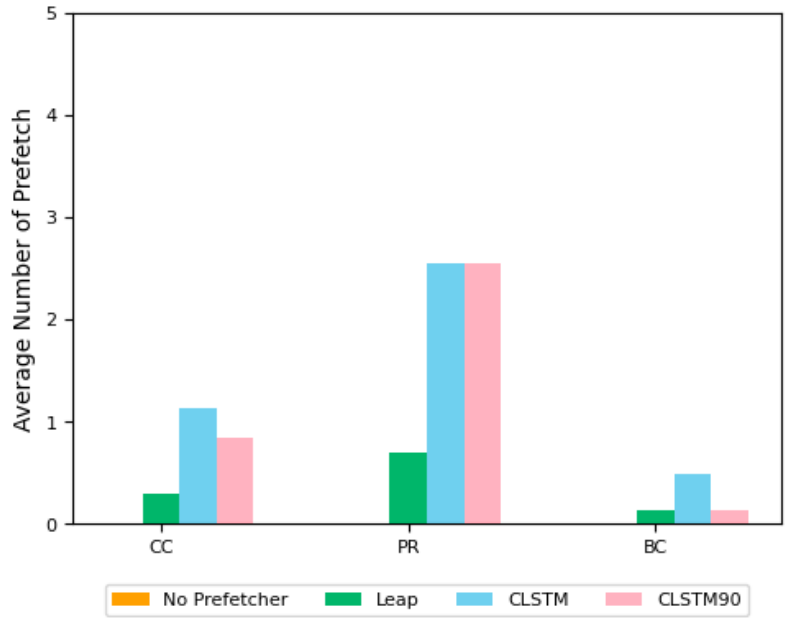


그림12. Trace에 따른 프리페치 Average Number of Prefetch

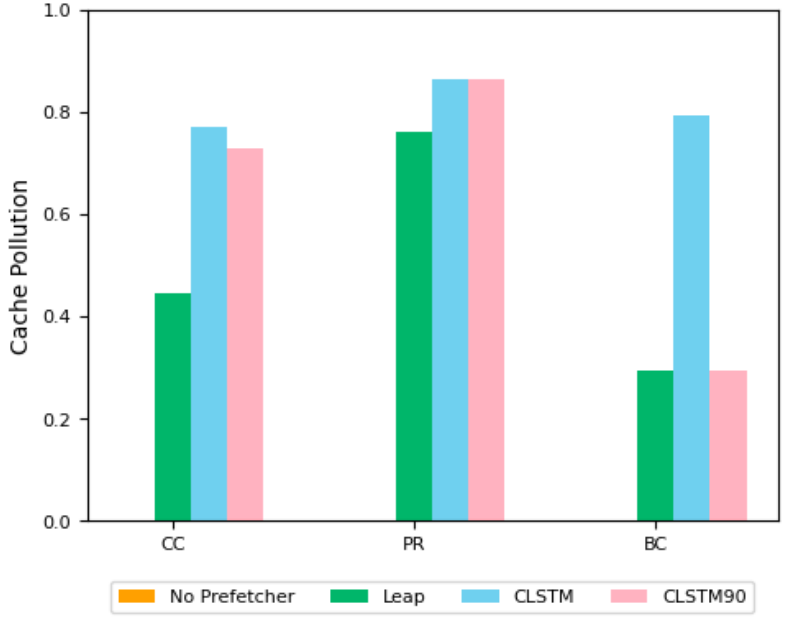


그림13. Trace에 따른 프리페치 Cache Pollution

결론

- CLSTM 적용 시,
  - ✓ 기존 프리페치 기법 대비 약 10% 성능 향상
  - ✓ SSD 속도로 인한 성능 저하 개선 위해 공격적 프리페치 적용
  - ✓ 기존 프리페치 대비 자원 사용률 ↑ (Cache Pollution, Bandwidth 낭비)

- CLSTM90 적용 시,
  - ✓ CLSTM과 비슷한 수준의 Hit Ratio
  - ✓ CLSTM의 Cache Pollution과 Bandwidth 낭비를 최소화 할 수 있는 방법
  - ✓ 여전히 개선 사항 존재

향후 발전 방향

- 두 가지 이슈 존재
  - ✓ Timeliness
    - Online 동작
    - 프로그램 실행 중, 실시간 학습데이터 처리 & 학습 & 추론
  - ✓ Granularity mismatch
    - I/O 처리 과정에서의 Bandwidth 낭비, Cache Pollution 추가 개선

개발 추진 현황

날짜	목표	수행여부
3주차 (~3/22)	아이디어 구체화	✓
4~5주차 (~4/5)	기능 구현 및 성능 측정	✓
6주차 (~4/12)	프로토타입 구현 및 중간 결과 정리	✓
7~8주차 (~4/26)	트레이스 추가 Top-K 설정	✓
9~10주차 (~5/10)	기능 개선 (동작 시점에 따른 버전)	✓
11~12주차 (~5/22)	논문 작성 및 최종 결과 정리	△

참고 문헌

[1] Leeor Peled, Shie Mannor, Uri Weiser, and Yoav Etsion. 2015. Semantic locality and context-based prefetching using reinforcement learning.
In Proceedings of the 42nd Annual International Symposium on Computer Architecture. 285–297

[2] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning memory access patterns. In International Conference on Machine Learning. PMLR, 1919–1928.

[3] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. 2021. A hierarchical neural model of data prefetching. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 861–873.

[4] Hasan Al Maruf and Mosharaf Chowdhury. 2020. Effectively prefetching remote memory with leap. In 2020 USENIX Annual Technical Conference (USENIX ATC 20). 843–857.

감사합니다