Deep Learning Model for Content Aware Caching at MEC Servers

Introduction

Introduction

- **MEC server**
  - **between the core network and mobile devices**
  - **Caching the popular content into the MEC server**
  - **Increasing the overall cache hit ratio.**

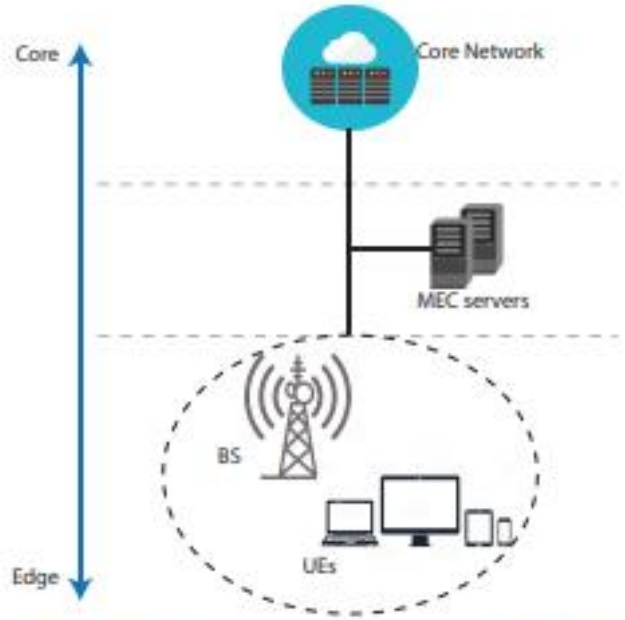


Fig. 1: Three-layer architecture of MEC

- **가장 기초적인 방법론 : 가장 많이 요청된 contents를 MEC server에 caching 하는 것.**

Introduction

- Caching Content에 대해 두 가지 전략 존재
  - Proactive Caching : predicts the popularity of content, caches the most frequently requested content.
    - even before the requests from the clients have arrived.
  - Reactive Caching : decision whether to cache or not only when the request for the given content arrives

- When User request contents, first searched in the MEC server.
  - If found (cache hit), the UE is directly able to fetch the requested video from the MEC server.
  - Otherwise, it is retrieved from the core network (cache miss).

- With frequent misses at the MEC server, the backhaul congestion(backhaul 혼잡) and access delay increases.

System overview

System overview

- System overview
  - EU(=End Users)
    - 사용이 끝난 유저들은 BS와 connect
  - CCS(Core Content Server)
    - BS와 backhaul로 연결되어 있음
  - MEC server
    - Run various AI-algorithm from EU devices
    - 세가지 모듈로 구성
      - CRM(Content Request Module)
      - CDM(Cache Decision Module)
      - CM(Cache Memory)

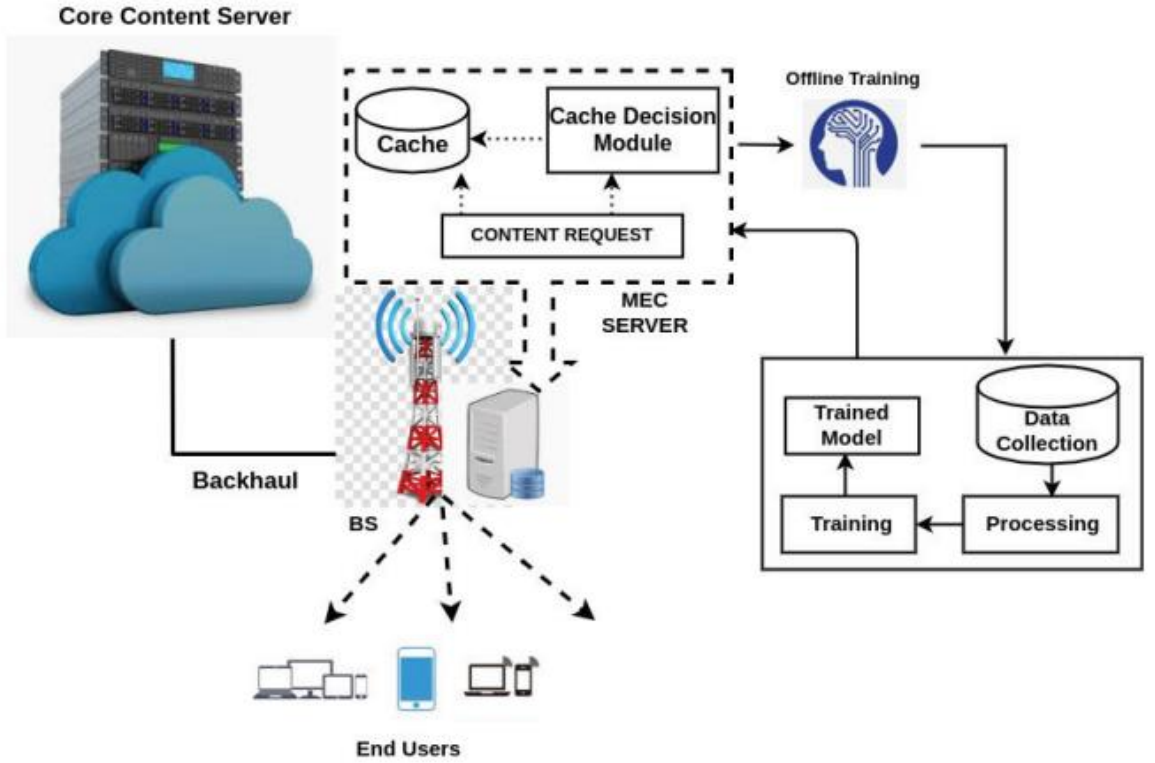


Fig. 2: Hypothetical Scenario of Content Aware Caching in MEC

System overview

- **MEC server**
  - **CRM(Content Request Module)**
  - **CDM(Cache Decision Module)**
  - **CM(Cache Memory)**

1. **CRM이 사용자로부터 Request 수신**
2. **요청 로그를 CDM으로 전송**
3. **MEC server에 있을 경우 사용자에게 전송**
4. **없을 경우 CCS에서 가져와 전송**
   1) **해당 요청 로그는 학습 데이터로 사용됨**

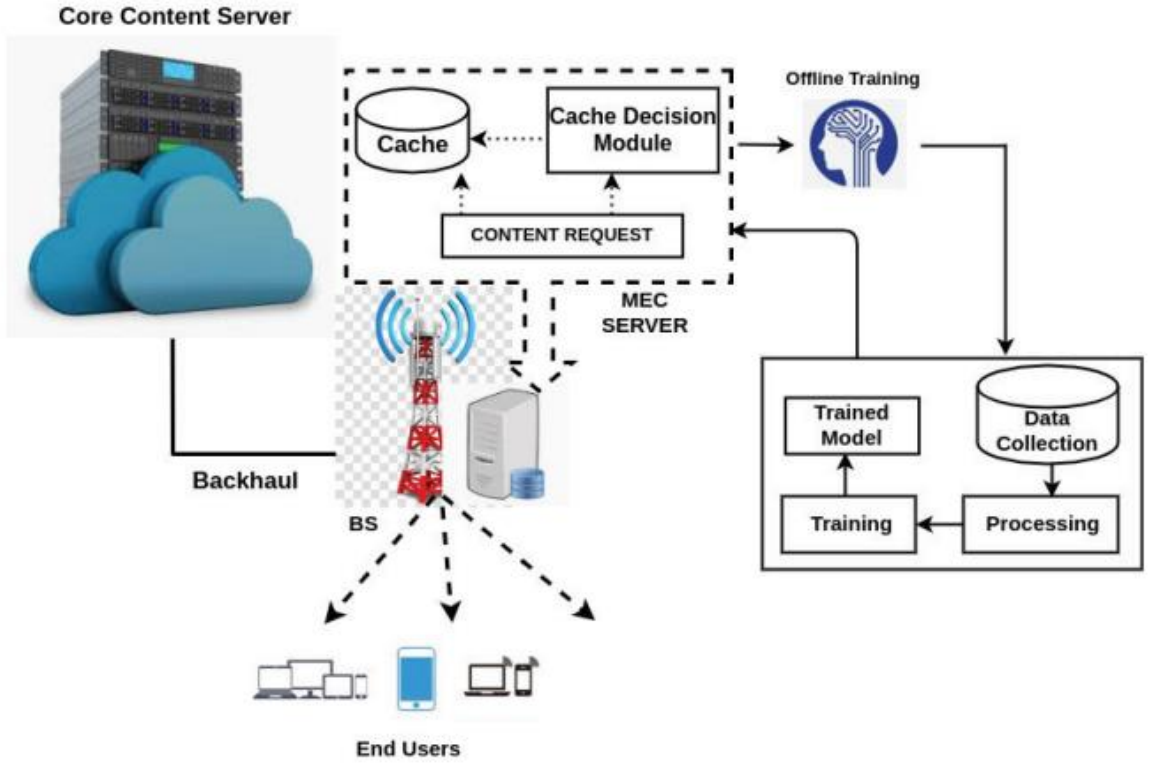


Fig. 2: Hypothetical Scenario of Content Aware Caching in MEC

System overview

- **Popularity of the movie may change over time**
  - Popularity can be defined locally or temporally
  - Prevalent in a specific time window of day.

- **예시 : family drama는 evening, Horror movie는 late night 에 선호된다.**
  - Observing such temporal characteristics of the popularity (of a movie), LSTM could be used to model the popularity, considering its efficacy to model the sequential data.

- **Using LSTM (example)**
  - Example : 영화 장르 예측
  - $\overrightarrow{SGV_{ij}} = f(\overrightarrow{SGV_{(i-1)j}}, \overrightarrow{SGV_{(i-2)j}}, \dots, \overrightarrow{SGV_{(i-7)j}})$
    - SGV : genre vector
  - Particular time (for 7 days) 를 고려하여 predict

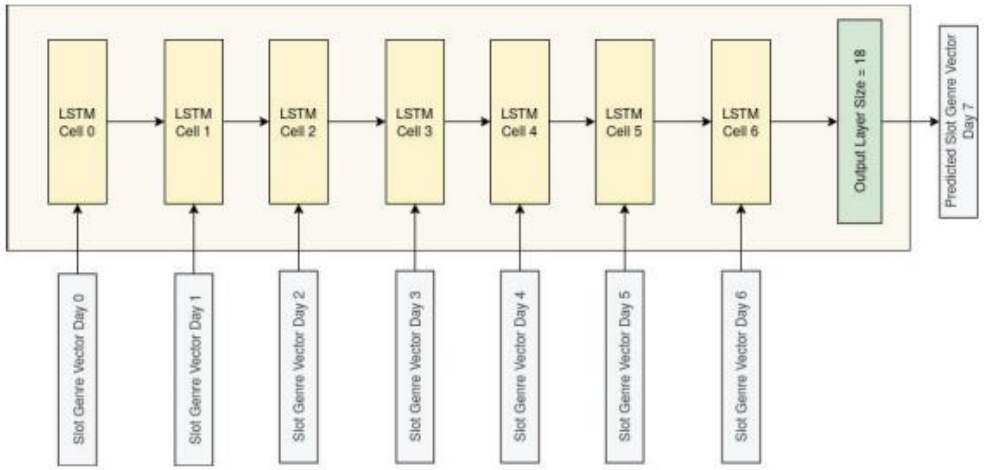


Fig. 3: Overview of LSTM for Genre Prediction

Model

Model

- **DCache model**
  - **uses a stream of video requests from various end-users as input data for caching the most popular video content in the MEC server.**
  - **Streaming services like Netflix, Amazon Prime Video, YouTube, etc.**

- **MovieLens dataset 사용**

- **Input Stream : 4 parameters**
  1. **User-ID : who requests the movie**
  2. **Movie-ID : An unique ID for every movie/video hosted by the OTT services**
  3. **Timestamp : The time when a given movie is rated**
  4. **Tag: Tags denotes ratings/number of views/comments given for a particular movie/video.**

Model

- **Data Preprocessing Module : consists of 2 modules**
  1. **Data Cleaner 1 : create genre vector $\overrightarrow{M}$ (features : userId, MovieID, timestamp, ratings)**
  2. **Data Cleaner 2 : combines the output of Model 1 and previous views of a given movie and fed into Model 2**

- **Slot-wise Genre Prediction Module(=Model 1)**
  - **the first step of the proposed two-step model for predicting the movies' popularity**
  - **Input : features prepared by Data Cleaner 1**
  - **Output(=predict) : most prevalent genre of content at six different time slots the day using LSTM architecture**
    - **하루를 6개의 time slot으로 나누어 해당 time slot 에서 가장 prevalent genre 를 predict**

- **Total Request Count Module(=Model 2)**
  - **predicts the total request count for a movie using a combined neural network comprising LSTM and Deep Neural Network (DNN)**
  - **Input : slot-wise predicted genre vector , the previous views of the movie for the last seven days processed by the Data Cleaner 2**

Model

- **Evaluator Module**
  - $\mathcal{L}\,(final\ list)$: $predicted\ views\ from\ the\ model\ 2$
  - **Popularity(인기도) 높은 순서로 cache가 가득 찰 때까지 cache.**

  - **측정 방법 : cache hit probability, backhaul usage, and the access time taken to retrieve a given movie.**

Problem formulation

Problem formulation

- $R = \{r_1, r_2, \dots, r_k\}$, r_k is the number of arriving requests at a BS
- $V = \{v_1, v_2, \dots, v_n\}$, sets of movies(N개의 movie = content)
  - r_{v_i} : $number\ of\ requesting\ v_i\ movie$

- If, r_{v_i} arrive → $check\ local\ cache$
  - Present : sent to the user without delay
  - No present : check CCS and sent to the user with a delay δ

- $K = capacity\ of\ each\ BS$(대문자 K)
  - k_{v_i} : $the\ size\ of\ each\ movie\ v_i$(소문자 k)
  - $\sum_{i=1}^{n} k_{v_i} \leq K$ → 모든 movie의 size(용량)의 합은 BS의 capacity보다 클 수 없다.

Problem formulation

- $x_{v_i}^t \in \{0, 1\}$, $decision\ variable$
  - Whether or not to cache a given movie v_i at time t at the MEC server
  - 0 : not cached
  - 1 : cached.
  - $\sum_{i=1}^{n} x_{v_i}^t k_{v_i} \leq K$

- Delay time : $y_{t+1}^{delay} = \sum_{i=1}^{n} \left((1 - x_{v_i}^t) \delta k_{v_i} \right) \hat{\phi}_{v_i}^{t+1}$
  - δ : latency for retrieving the movie v_i(영화 검색 대기 시간)
  - $\hat{\phi}_{v_i}^{t+1}$: predicted number of request for the movie v_i in the future time (t+1)
  - 목적 : minimized access time, maximizing the probability of cache hit at the MEC servers.

Problem formulation

- **Objective function : $\mathcal{P}_{hit}$**
  - $\mathcal{P}_{hit} = \frac{\sum_{t \in T} \sum_{i=1}^{n} x_{v_i}^t r_{v_i}^t}{\sum_{t \in T} \sum_{i=1}^{n} r_{v_i}^t}$, subject to $\sum_{i=1}^{n} x_{v_i}^t k_{v_i} \leq K$
  - Optimal solution 을 위해서는 $\hat{\phi}_{v_i}^{t+1}$이 필요 → two deep learning model(LSTM and DNN)을 사용하여 계산.

Architecture of proposed DCache model

Architecture of proposed DCache model

1. Data Pre_processing :

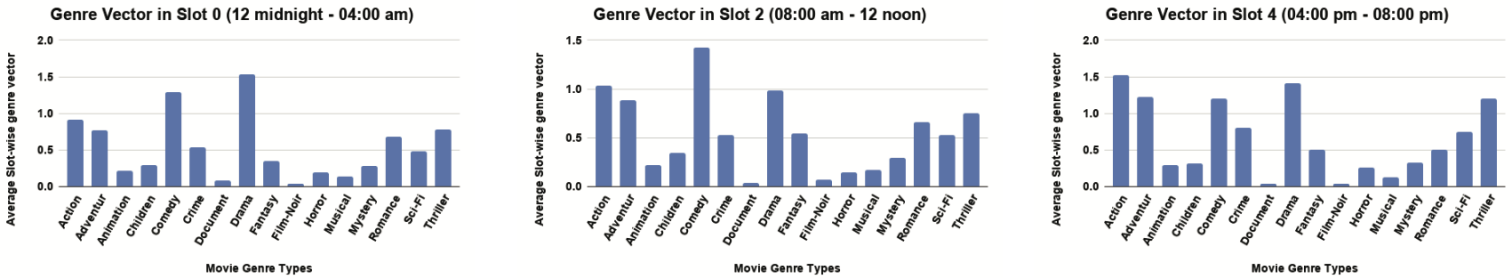
- **Content popularity :** $\frac{the\ number\ of\ requests\ received\ for\ a\ movie}{total\ number\ of\ requests\ obtained\ for\ a\ given\ period.}$
- $Content\ popularity(F_{v_i}) = \frac{z_{v_i}}{\sum_{i=1}^{N_{s0}} z_{v_i}}$,
  - z_{v_i} : $total\ number\ of\ views\ for\ a\ movie\ v_i\ in\ a\ slot\ s_0$
  - N_{s_0}: $the\ number\ of\ total\ requests\ in\ a\ slot\ s_0$
- **Movie genre vector($\overrightarrow{M}$) :** $<'Action','Adventrue',\dots,'War','Western'>$ **와 같이 18개 class에 대한 vector.**
  - **Value 1 : belongs to a particular genre**
  - **Value 0 : not belongs to a particular genre**
- **Time slots : 6개로 구분(4시간 씩)** $\rightarrow < s_0, s_1, s_2, s_3, s_4, s_5 >$
  - **Consider the changes in viewing trends of the users.**

Architecture of proposed DCache model

1. Data Pre_processing :

- Given slot 에서 prevalent genre 계산하기 위해 weighted average slot-wise genre vector($G_{avg}^{s_0}$) 생성

- $G_{avg}^{s_0} = \frac{\sum_{i=1}^{N_{s_0}} \vec{M}_{v_i}^{s_0} z_{v_i}}{\sum_{i=1}^{N_{s_0}} z_{v_i}}$
  - z_{v_i}(상수): $total\ number\ of\ views\ for\ a\ movie\ v_i\ in\ a\ slot\ s_0$
  - 따라서 $G_{avg}^{s_0}$: 평균 genre (request movie in a slot s_0)



a: Genre vector from 12 midnight to 4 am b: Genre vector from 8 am to 12 noon c: Genre vector from 4 pm to 8 am

Fig. 5: Statistical Analysis of Slot-wise breakdown of Genre Vector at different time slots of a day

시간대에 따라 prevalent genre가 달라짐을 확인할 수 있다.

Architecture of proposed DCache model

2. Slot-wise Genre prediction model using LSTM(=Model 1)

- To predict the slot-wise Genre Vector $\widehat{O}_{t+1}^{s_i}$ of future slots.
- Genre vector of the last seven days(일주일) → $window\ size\ to\ be\ seven$.
  - $D(Day) = < D_0, D_1, D_2, D_3, D_4, D_5, D_6 >$
- Input feature $X_t : D_t\vec{G}_{avg}^{s_i}$, $i \in \{0, 1, 2, 3, 4, 5\}$
  - i : 4hours , divided each day into six slots.

- Model 1(X_t) → $output$: $\widehat{O}_{t+1}^{s_i}$(slot − wise genre vectors of future slots)
  - Example : D_7 의 s_4 $slot$(PM 4~8)의 genre $vector$ 예측시
    - LSTM model 은D_0 부터 D_6 의s_4 $genre\ vectors$를 사용.

- Loss function : MSE

Architecture of proposed DCache model

2. Slot-wise Genre prediction model using LSTM(=Model 1)

- 정리 : input 으로 average weighted genre vector.
  - 해당 input은 지난 7일 간의 데이터, each day 는 6개의 time slots 으로 나누어져있음.
- 추론 : 미래(t+1) 시점의 genre vector 를 예측.

3. Total Request Count prediction model using LSTM+DNN(=Model 2)

- To predict the number of request : $\widehat{\phi}_{v_i}^{t+1}$ (for each movie in a given slot)
- Two different type of inputs
  - Sequential input(순차적인 input)
  - Instantaneous input(동시에 발생하는 input)

Architecture of proposed DCache model

3. Total Request Count prediction model using LSTM+DNN(=Model 2)

- **Two different type of inputs**
  - **Sequential input(순차적인 input) : consider a previous views $\overrightarrow{PV_{kij}}$, $\left(k^{th}\ video, ij^{th}\ slot\right)$**
    - **i : number of slots**
    - **j : number of days**
    - **example : $\overrightarrow{PV}_{k0j}$($k$ 번째 video, 0번째 slot, j 번째 day) = $<PV_{k0(j-1)}, PV_{k0(j-2)}, \cdots, PV_{k0(j-7)}>$**
    - **Consider a window size(7 days) and $\overrightarrow{PV}$ fed into the LSTM.**
  - **Instantaneous input(동시에 발생하는 input) : $\overrightarrow{M}$(movie genre vector) , $\widehat{O}_{t+1}^{s_i}$(predicted slot-wise genre vector)**
    - **Fed into a DNN**
- **Model_2($\overrightarrow{PV_{kij}}$ to LSTM , $\overrightarrow{M}$ and $\widehat{O}_{t+1}^{s_i}$ to DNN)**
  - **Outputs are combined using a merge net**
  - **Five more hidden layers are used between the merge net and the output layer.**
- **Loss function : Huber Loss**
- **Optimizer : Adam**

Architecture of proposed DCache model

- **Algorithm 1: DCache for Content Popularity Prediction**
  - **Input : Slot-wise genre vector of movies : $\vec{G}^{s}_{avg}$, Movie Vector : $\vec{M}$, Views in the last Seven days : $\overrightarrow{PV}_{kij}$**
  - **Output: Predicted request counts ϕˆt+1 vi**

1. **Request Arrival : R = {req1(v_i) , req2 (v_i) , … , reqk (v_i) }**
2. **Train Model 1 (m^*_1) (LSTM) predicting the slot-wise Genre Vector;**
3. **m^*_1.train();**
4. **for slots s ← 0 to 5 do**
5. **for data $m_1 \leftarrow \vec{G}^{s}_{avg}$ do**
6. **optimizer ← Adam;**
7. **$\hat{y} = m^*_1$(data m_1);**
8. **loss = MSE(y, $\hat{y}$);**
9. **loss.backward() ▷ Back Propagation ;**
10. **optimizer.step() ▷ Updates the parameters;**
11. **end**
12. **end**
13. **Train Model 2 (m^*_2) (LSTM+DNN) predicting views of movies;**
14. **m^*_2.train() ;**
15. **for data $m_2 \leftarrow \hat{O}^{s}_{t+1}$, $\vec{M}$, $\overrightarrow{PV}_{kij}$ do**
16. **optimizer ← Adam;**
17. **$\hat{y} = m^*_2$(data m_2);**
18. **loss = SmoothL1(y, $\hat{y}$);**
19. **loss.backward() ▷ Back Propagation ;**
20. **optimizer.step() ▷ Updates the parameters;**
21. **end**
22. **Finally store the predicted request counts $\phi^{t+1}_{v_i}$ in a list L (즉 미래 t+1 시점에서 v_i 의 request count를 예측)**

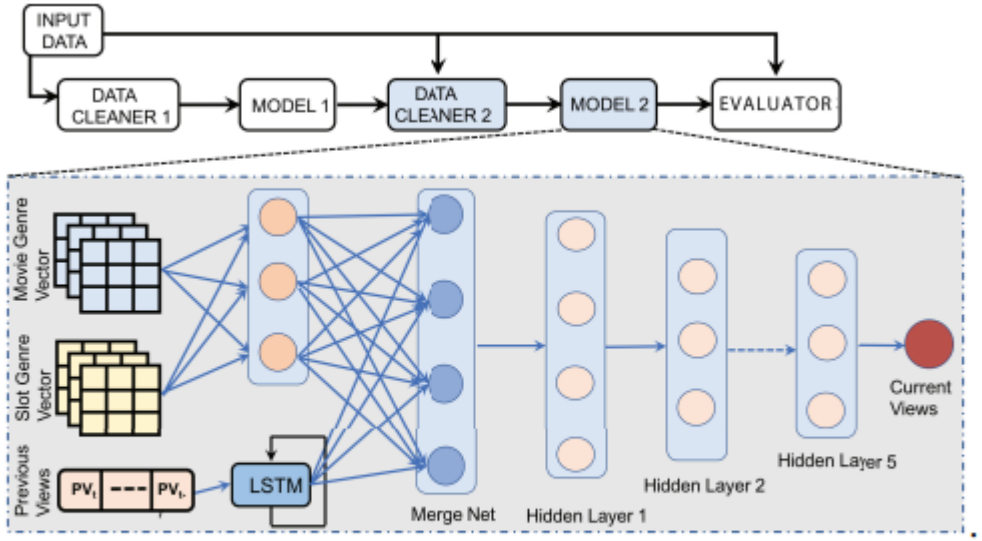


Fig. 7: Proposed Architecture for Total Request count Prediction Model

Architecture of proposed DCache model

4. Caching Decision and the Evaluator Module

- Model m_2^* predicts the final list of predicted views($\mathcal{L}$).
- $\mathcal{L} = \{l_{s_0}, l_{s_1}, l_{s_2}, l_{s_3}, l_{s_4}, l_{s_5}\}$ → 미래 t+1 시점에서 v_i 의 request counts
  - Pareto-principle : 20% of the most popular movie receives 80% of the request.
  - 따라서 상위 20프로 content(movie)만 저장하는 것이 효율적 → Consider 20% highest popularity movies
  - $C_v^{s_j} = 0.2 * l_{s_j}$, j ∈ $\{0, 1, 2, 3, 4, 5\}$

- Time slot(s) 마다 상위 20프로 contents(=movies)가 popularity 순으로 caching. (until the cache space is full)
- Time slot 이동 시, previous slot에 caching 되어 있던 contents 모두 삭제.
- 삭제 이후 다시 caching 진행.

Architecture of proposed DCache model

4. Caching Decision and the Evaluator Module

Algorithm 2: DCache for Caching Strategy

1. $\mathcal{L} = \{l_{s_0}, l_{s_1}, l_{s_2}, l_{s_3}, l_{s_4}, l_{s_5}\}$ slot-wise predicted list of movies;
2. for j = 0, 1, 2, 3, 4, 5 do
3. $C_{v_i}^{s_j} = 0.2 * l_{s_j}$;
4. $C_{v_i}^{s_j}$ consists of predicted list in sorted order;
5. while $C_{v_i}^{s_j}$ do
6. if $\sum_{i=1}^{n} k_{v_i}^{l_{sj}} \leq K$ then (cache space is not full)
7. Cache ← Cache + $C_{v_i}^{s_j}$;
8. end
9. end
10. while $req_{v_i}^{sj}$ do
11. if v_i present in $C_{v_i}^{s_j}$ then
12. MEC Cache → v_i ;
13. Calculate P_{hit} , $P_{backhaul}$, $Delay$
14. end
15. else
16. Main Content Server → v_i ;
17. Calculate P_{hit} , $P_{backhaul}$, $Delay$
18. end
19. end
20. Cache ←− Cache - $C_V^{s_j}$ ▷ Delete Cache of slot s_j (Delete all cached contents);
21 go to 3 line
22. end

EXPERIMENTS AND RESULTS

EXPERIMENTS AND RESULTS

- **MovieLens Dataset**
  - **45,000 movies**
  - **26,000,000 ratings done by 270,000 users.**
  - **User information : age, sex, occupation**
  - **Movie information : movie-ID, genre, year, day of the year, month and day**

- **Slot-wise genre format : used the movies and ratings file**
  - **Movies file : < movie-ID, title, genres >**
  - **Ratings file : < user-ID, movie-ID, tag, timestamp >**

- **Slot-wise genre vector : $\vec{G}_v^s$**
  - **v : content 개수(=the number of genres)**
    - **MovieLens Dataset : 18개의 genre 존재**
  - **s : time slot(하루를 4시간씩 등분 0,1,2,3,4,5)**

EXPERIMENTS AND RESULTS

❑ Experimental Setup

- **Pytorch**
- **DGX workstation(Tesla V100 GPU, 5120 Cuda cores and a GPU memory of 32GB)**
- **Also, test in a CPU server with 128GB CPU memory**

❑ Evaluation Metrics

- **Probability of Hit P_{hit} : the probability that the requested movie is present in the local cache of the MEC server**
  - **If the requested movie is present in the cache, it is a hit**
  - **Else, it is considered to be a miss**

- $\mathcal{P}_{hit} = \frac{\sum_{t \in T} \sum_{i=1}^{n} x_{v_i}^t r_{v_i}^t}{\sum_{t \in T} \sum_{i=1}^{n} r_{v_i}^t}$ **, subject to** $\sum_{i=1}^{n} x_{v_i}^t k_{v_i} \leq K$
  - k_{v_i} : *the size of each movie* v_i(소문자 k)
  - $x_{v_i}^t \in \{0, 1\}$, *decision variable* (cache or not)
  - $r_{v_i}^t$ **is the number of request for the movie** v_i **at the time t**

EXPERIMENTS AND RESULTS

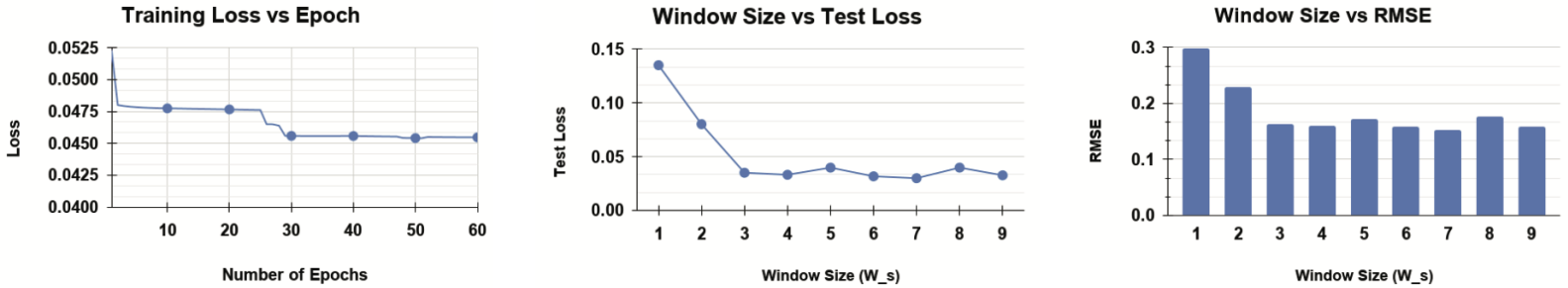
❑ Evaluation Metrics

- $P_{backhaul}$: calculates the network traffic in the backhaul links between the core network and the MEC server
  - $P_{backhaul} = \sum_{t \in T} \sum_{i=1}^{n} k_{v_i} \left(1 - x_{v_i}^t\right) r_{v_i}^t$
    - k_{v_i} : *the size of each movie* v_i(소문자 k)
    - $x_{v_i}^t \in \{0, 1\}$, *decision variable* (cache or not)
    - $r_{v_i}^t$ is the number of request for the movie v_i at the time t
  - Measures congestion

- Delay time : $y_{t+1}^{delay} = \sum_{i=1}^{n} \left((1 - x_{v_i}^t) \delta k_{v_i}\right) \hat{\phi}_{v_i}^{t+1}$
  - δ : latency for retrieving the movie v_i(영화 검색 대기 시간)
  - $\hat{\phi}_{v_i}^{t+1}$: predicted number of request for the movie v_i in the future time (t+1)
  - 목적 : minimized access time, maximizing the probability of cache hit at the MEC servers.

EXPERIMENTS AND RESULTS

- **Evaluation of slot-wise Genre Prediction Model(Model_1)**
  - $Model_1$: **predict the most prevalent genre in aparticular time slot of the day**
  - **Evaluation metrics : RMSE**



a: Train Loss Vs Epoch b: Window Size vs Test Loss c: Window Size vs RMSE

Fig. 8: Performance of Slot-wise Genre Prediction Model m_1^*

TABLE V: Comparison of RMSE for Model 1

Model	Window size ($W_s = 3$)		Window size ($W_s = 7$)		Window size ($W_s = 9$)	
	Test Loss	RMSE	Test Loss	RMSE	Test Loss	RMSE
RNN (m_1^*)	0.1121	0.28595	0.03151	0.1599	**0.0310**	**0.15688**
LSTM (m_1^*)	**0.0350**	**0.16339**	**0.03000**	**0.1523**	0.03169	0.15710

EXPERIMENTS AND RESULTS

- **Evaluation of Request Count Prediction Model(Model_2)**
  - $Model_2$: predicts the list of views for all the movies in various time slots of the day
  - 다른 DNN 모델과 비교해보았을 때, Model 2 는 historical relationship capture 가능.

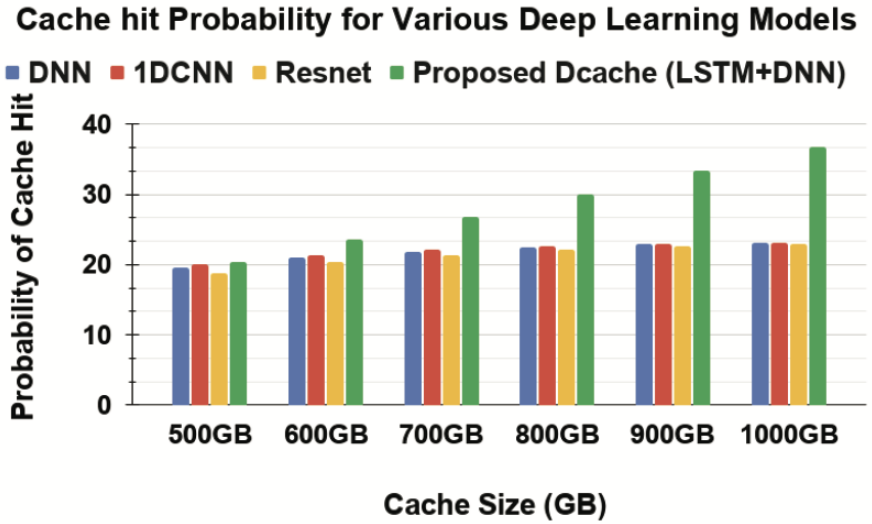


Fig. 9: Probability of Cache Hit for Various Deep Learning Models

TABLE VI: Cache Hit Probability against MEC Cache Size (GB)

Models	Cache Hit Against MEC Cache Size (GB)					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
DNN	15.9	18.78	21.59	24.33	26.98	29.6
1DCNN	16.16	19.09	21.9	24.65	27.29	29.85
Resnet	15.94	18.85	21.71	24.47	27.13	29.76
Proposed (LSTM+DNN)	**20.38**	**23.52**	**26.72**	**30.07**	**33.41**	**36.82**

EXPERIMENTS AND RESULTS

❑ Evaluation of Proposed DCahe Caching Strategy

- **Final predict : List $\mathcal{L} = \{l_{s_0}, l_{s_1}, l_{s_2}, l_{s_3}, l_{s_4}, l_{s_5}\}$**
  - **l_{s_0} : s_0 (00:00 – 04:00) time slot에서 예측된 movies 요청 수**
    - **l_{s_0} : < movie 1, movie 2, … > → consists of the request in descending order of content popularity in a slot-wise manner.**

TABLE VII: Cache Hit Probability against MEC Cache Size (GB)

Models	Cache Hit Against MEC Cache Size (GB)					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
LRU	19.553	20.96	21.87	22.46	22.86	23.12
LFU	20.003	21.26	22.09	22.62	22.93	23.15
FIFO	18.82	20.34	21.39	22.11	22.6	22.93
DeepMEC [14]	15.23	18.56	20.89	24.01	28.11	30.4
DeepCache [15]	16.16	17.15	17.8	18.2	18.48	19.1
RL-Cache [16]	17.01	20.37	23.65	26.74	29.69	32.44
Proposed *DCache*	**20.38**	**23.52**	**26.72**	**30.07**	**33.41**	**36.82**

EXPERIMENTS AND RESULTS

❑ Evaluation of Proposed DCahe Caching Strategy

▪ Backheaul Usage

- **If request is not present in the MEC cache, increasing backhaul traffic**
- **The better the cache hit probability, the lesser the network congestion at the backhaul links.**

TABLE VIII: Computation of Slot-wise Backhaul Usage (TB)

Slots	**Slot 0**	**Slot 1**	**Slot 2**	**Slot 3**	**Slot 4**	**Slot 5**
Backhaul Usage (TB)	301.64	235.68	170.6	160.19	182.422	240.94

TABLE IX: Backhaul Usage (TB) against MEC Cache Size (GB)

Models	Backhaul Usage (TB) Against MEC Cache Size					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
LRU	1691.5	1661.08	1642.59	1630.19	1621.91	1616.45
LFU	1682.03	1655.58	1638.14	1627.23	1620.32	1615.86
FIFO	1706.76	1674.76	1652.83	1637.72	1627.39	1620.69
DeepMEC[14]	1764.90	1723.99	1661.09	1576.87	1512.03	1448.85
DeepCache [15]	1710.35	1759.76	1747.24	1738.42	1732.46	1720.8
RL-Cache [16]	1690.2	1660.15	1610.22	1465	1416	1361.01
Proposed *DCache*	**1673.96**	**1607.94**	**1540.82**	**1471.68**	**1399.98**	**1328.39**

EXPERIMENTS AND RESULTS

❑ Evaluation of Proposed DCache Caching Strategy

- **Access Delay : The access delay further reduces when the cache size of the MEC is further increased**
- **Nowadays, most streaming solutions use HTTP-based adaptive streaming mechanisms with a slightly higher latency range between 15-45 seconds.**

TABLE X: Access Delay (ms) against MEC Cache Size (GB)

Models	Access Delay (sec) Against MEC Cache Size					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
LRU	17.32	17.016	16.82	16.69	16.60	16.55
LFU	17.22	16.95	16.77	16.66	16.59	16.54
FIFO	17.47	17.14	16.92	16.77	16.66	16.59
DeepMEC[14]	18.29	17.65	17.00	16.14	15.48	14.83
DeepCache [15]	18.04	17.83	17.7	17.62	17.55	17.41
RL-Cache [16]	17.86	17.04	16.43	15.78	15.13	14.54
Proposed *DCache*	**17.14**	**16.46**	**15.77**	**15.07**	**14.33**	**13.6**

- **Consider the cache size on the higher side due to two main reasons**
  - **Firstly, every four hours we replace the cached movie in a slot-wise manner.**
  - **Secondly, MECs have very high storage and computational capacities. Therefore, our proposed model DCache provides a higher cache hit probability with lower backhaul usage and access delay while considering the MEC cache's size as a constraint.**

EXPERIMENTS AND RESULTS

- **Evaluation of Proposed DCache Caching Strategy**
  - **Training Cost : take time**

TABLE XI: Computation of training cost **in minutes** for Various Deep Learning Models

Schemes	80 K Iteration		160 K Iteration		240 K Iteration		320 K Iteration	
	Time (min)	Train Loss	Time (min)	Train Loss	Time (min)	Train Loss	Time (min)	Train Loss
DNN	17.4	0.86	40.03	0.841	53.75	0.834	74.45	0.793
1DCNN	46.24	0.56	92.66	0.549	137.42	0.544	182.8	0.515
RESNET	53.16	0.40	107.01	0.35	160.48	0.331	217.12	0.29
Proposed DCache	21.84	0.0611	47.82	0.031	74.02	0.021	100.84	0.016

DISCUSSION

DISCUSSION

- **Only when we have information about the video id and the number of times it is being requested.**
  - **Need video context data**

- **DCache proposed an efficient caching mechanism at the MEC server based on the popularity of a movie at different time slots of a day.**

CONCLUSION

CONCLUSION

- DCache for performing caching at the MEC server based on the content popularity where each day is divided into slots of prevalent genres of movies.
  - The cache present in the MEC server immediately replies with the requested movie to the end-users if present in the cache.
  - Else it fetches the movie from the central content server.

- DCache outperforms standard caching mechanism like LRU, LFU, FIFO, DeepMEC, DeepCache and RL-Cache with respect to cache hit probability, backhaul usage and access delay with a constraint on the cache size

- Next
  - AoI-based Temporal Attention Graph Neural Network for Popularity Prediction and Content Caching