IoTCache: Toward Data-Driven Network Caching for Internet of Things

***Abstract*—This paper proposes IoTCache, a popularity based caching solution for IoT. Our work mainly consists of three parts: Firstly, we build a large dataset of real popularity of IoT data. By analyzing the popularity features observing from the dataset, we derive the popularity distribution and further propose the Popularity Evolving Model (PEM) to model the IoT data's popularity changing. Secondly, we design a data-driven method to generate high accurate popularity model by using an elaborate designed Deep Neural Network. We further design a simple but high efficient prediction method called statistic based PEM to deal with cold boot problem. Thirdly, based on the model generated, we design the popularity-based evicting and prefetching algorithms. We evaluate IoTCache in two developed IoT platforms based on Content Delivery Network and Named Data Network respectively. Experiment results show that IoTCache can significantly increase the cache hit ratio, and decrease the IoT edge traffic and data latency.**

I. Introduction

Internet of Things (IoT), as the trend of future networks, has become a hot research topic for both industry and academia recently [1], [2]. Billions of IoT devices generate tremendous amount of measurement, monitoring, and automation data while a large number of end-users around the world running applications consume these data. Although each IoT traffic is commonly regarded as be a low rate,[1] the aggregate load on core networks is expected to be large. Therefore, it is important to reduce traffic, lower communication latency, and reduce data communication overheads for network providers.

On the other hand, in many IoT applications, a lot of end-users likely to request similar IoT data, such as air quality and traffic conditions of hot areas. Similar to the Internet traffic, a large percentage of network traffic is redundant-multiple users request much of the same content. It is natural to exploit caching technology to reduce the redundancy caused by delivering similar content over the network.

Network caching, keeping frequently accessed content in a location close to the requester, is extensively studied and used in the Internet [3]. There are two main routes: 1) Edge caching – the contents are cached in the edge servers that are close to the end-user at the edge of the network (Content Delivery Network, CDN [4], [5], is a representative); 2) In-network caching – the contents are cached in routers of the core network, i.e., caching is considered as a basic network functionality (Information Centric Network, ICN [6], is a representative). Following the two routes, some researchers start to explore the IoT caching problem in recent years.

- *Edge caching for IoT*. On one hand, many recent works discuss the caching issues of edge computing [7] or 5G mobile communication [8], [9] for IoT scenarios. On the other hand, some traditional CDN providers also provide specific solutions for IoT, such as Akamais "Over the Air (OTA) solution [10] and Microsoft Azure CDN [11].
- *In-network caching for IoT*. Paper [12] is the first study on caching IoT data in Internet content routers. After that some works [13], [14] design caching strategies for IoT which exploit the ICN architecture.

For both edge caching and in-network caching, designing an efficient content caching system should answer the questions that which content should be cached, where and when. Existing work on the Internet [15] points out traditional cache replacement algorithms, such as Least Recently Used (LRU), Least Frequently Used (LFU), or their variants, may suffer major performance degradation since they ignore the popularity that a content may request, which may alter the future network traffic pattern. i.e., it is desired to incorporate the future popularity of content into the caching decision making. However, many fundamental questions about IoT data popularity and related popularity based caching have no answer until now.

This paper tries to give answers for these questions, and then proposes IoTCache, a popularity based caching solution for IoT. There are two main challenges:

1) What is the popularity pattern of IoT data and how to model it? Popularity is considered as the critical influence factor in cache systems. Therefore, we collect the real popularity of IoT data from Internet and build a large popularity dataset (about 27 regions and 501400+ records). Base on the analysis on the dataset, we find three main features of IoT data's popularity and reveal their significant influence in caching mechanism design. More important, we find that the classical traffic model (such as: Independent Reference Model (IRM) [16], Shot Noise Model (SNM) [17], etc.) may lose efficacy in this scenario. Therefore, we derive the request process and popularity distribution of IoT data, and further propose a Popularity Evolving Model (PEM) that can demonstrate the popularity changing of the IoT data. The proposed PEM helps to understand the behavior of IoT data's popularity, and benefit to the cache system/strategy designing.

2) How to predict IoT data's popularity and design appropriate cache algorithms? Even we can model the revealed popularity via PEM, but how to predict the future popularity is still a key problem because of the context-related, non-linear and un-continuous popularity changing process and massive fragmented, diverse types and timeliness data in IoT environment. Thus, we design a Deep Neural Network (DNN) [18] learning core called EML: an embedding multi-layer Long Short-Term Memory (LSTM) [19] to train hight accurate PEM. Moreover, we design a simple but efficient modelling method called statistic based PEM to deal with cold boot. By using the generated PEM as the popularity predictor, we further design a popularity-based evicting and prefetching algorithm to address "what to cache" and "when to cache" problems.

[1]In fact, more and more IoT devices equipped multimedia sensors, e.g. cameras and micro-phones, which generate streaming data with high rate.

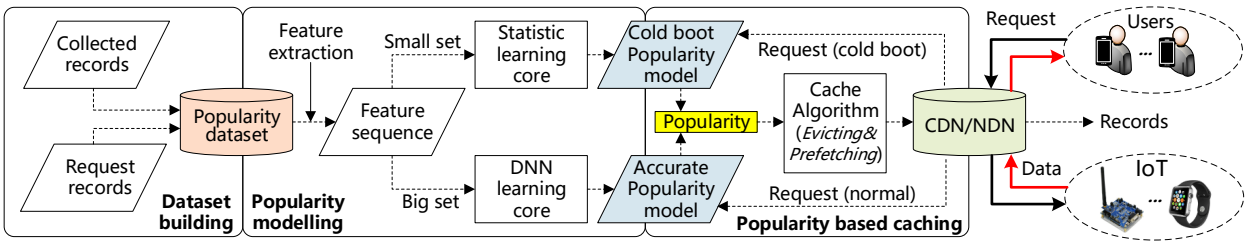


Fig. 1: Architecture of IoTCache.

To summarize, we make three contributions as follows:

- We build a large popularity dataset based on the real popularity of IoT data. Base on the analysis of the dataset, we find important features and derive the distribution of IoT data's popularity. Further, we propose the Popularity Evolving Model (PEM) that can model the popularity changing of IoT data and help design the cache system.
- We propose the data-driven popularity prediction method, which consists of two parts: DNN based PEM and statistic based PEM. The former uses an elaborate designed DNN that can generate high accurate popularity model of IoT data. The latter is can be used to deal with cold boot problem when the cache system begins to work.
- We design the popularity-based evicting and prefetching algorithms, and deploy them into two developed IoT platforms based on CDN and Named Data Networking (NDN) [20] respectively. We evaluate IoTCache in these platforms. Experiment results show that IoTCache can significantly improve the cache hit ratio and gain great reduction in IoT edge traffic and data latency.

II. Related work

The existing works about caching system for IoT can be divided into two categories:

The first category is Edge caching. Many recent works discuss the caching issues of edge computing or 5G environment for IoT scenarios. For example, Paper [8] exploits and advances the extreme edge 3C (communication, computing, caching) paradigms toward enabling the 5G ecosystem to meet its own criterion for low end-to-end latency and high Qos/QoE. Similarly, Paper [9] proposes the edge caching scheme for 5G mobile network based on the evolved packet core network caching and radio access network caching. Paper [7] surveys the convergence of 3C solution on mobile edge network. Moreover, the ubiquitous caching idea also be considered in 4G/5G network to decrease the backhaul traffic by using cache in serving gateway (S-GW), packet data network gateway (P-GW), and mobility management entity (MME), etc [9].

The second category is In-network caching. Paper [12] provides a method to determine whether a given data item should be cached according to the items lifetime, the rate and time range of incoming requests, and router hop distances to the data source and the end users. Paper [13] proposes a multi-attribute caching decision algorithm in CCN-IoT network by considering the content store size, hop count, particularly key temporal properties (data freshness and the node energy level). Paper [14] proposes a collaborative caching strategy for ICN-based wireless sensor networks by allocating and shifting the cache tasks according to nodes' ability, location and importance so as to adapt for the resource-constrained IoT environment.

However, these techniques do not consider the heterogeneity in IoT and the popularity features of IoT data, and make them may not perform well in IoT environment.

Our work also relates to the prediction of popularity and popularity based evicting/prefetching algorithm. For example, Paper [15] introduces a learning method based on the short term popularity. However, this method is based on the short term history data, which cannot provide a full changing pattern of the popularity. In fact, all the caching strategies (e.g., LFU) based on the history request rate cannot reach a good performance in IoT environment. Because these methods lack of consideration in the saltation and timeliness of IoT data's popularity. In contrast, Paper [21] takes small samples' popularity into account and tries to learn the popularity features more accurately by a global cache. However, this method is based on the SNM model thus may lose efficacy in IoT. Our method is different from these methods due to the consideration of the data and environment features in IoT, and gain the improvement compared with traditional methods.

III. Overview

In this paper, we propose a cache system named IoTCache for IoT environment to achieve two main goals: 1) Minimizing the data latency; 2) Reducing the duplication transmission. Fig. 1 illustrates the system architecture of IoTCache, which mainly consists of three parts: dataset building, offline learning and real-time caching.

Dataset building. In order to investigate the features of IoT data's popularity, a large dataset of IoT data's popularity is needed. Unfortunately, due to the closed characteristic of most IoT system and consideration in economy, security, privacy, etc., this type of dataset is rare and hard to obtain. Therefore, We use more than 9 months to build such dataset by collecting the popularity from Internet (see Sec. IV-A). Finally, we build the popularity dataset containing over 501400 records and more than 27 regions. Based on this dataset, we analyze the features and distribution of IoT data's popularity (see Sec. IV-B and IV-C), and further give the Popularity Evolving Model to illustrate the popularity changing process(see Sec. V). We note that when use IoTCache in the specific environment, the popularity dataset and related PEM should be based on the historical request traces of the environment.

Popularity modelling. In this part, IoTCache utilizes the dataset built to generate the popularity model. For the first, IoTCache extracts the records as the feature sequences that use to feed the learning core. Specifically, the learning core consists of two parts: 1) Statistic learning core, which is used to generate the model when system does not have enough dataset at beginning. The learning method of this core is a so-called Statistic PEM (SPEM, see Sec. V-A). This method is simple but high efficient that uses only few data and low computation resource so that is very useful to deal with cold

boot. 2) DNN learning core, which is used to generate accurate model when system has big history dataset. This core uses an elaborate designed DNN called Embedding Multi-layer LSTM (EML, see Sec. V-B). This core needs much time in offline training and large dataset to feed, but can provide high accurate prediction results.

Popularity based caching. In this part, IoTCache uses the model generated to predict the popularity of an IoT data. Based on the prediction value, the cache node runs the cache algorithms(see Sec.VI-A) to make the cache decision. Specifically, the cache algorithm consists of two parts: 1) Popularity-based evicting algorithm that decides which data should be evicted from the cache to improve the cache hit ratio, and 2) Popularity-based prefetching algorithm that decides which data should be prefetched to further decrease the latency.

IV. Popularity Analysis of IoT Data

A. Dataset Building

As mentioned before, we build a large dataset of IoT data's popularity. Table I illustrates the details of the dataset build. We collected the data from two searching engines: *Google Trends*[2]*(GT)* and *Baidu Index*[3]*(BI)* by downloading the searching popularity of overall 7 types of specific words (temperature, humidity, PM2.5, traffic, AQI, light, weather). Specifically, we collected the popularity for each hour and divide them in two levels: city level (12 cities, e.g., New York, Beijing, etc) and district level (15 districts, e.g. Brooklyn, Queens, etc). Then, we extracted the features of the time, interested area, types, searching location, etc, and combined the corresponding popularity value as the label to build the items in the dataset. We built the dataset from Nov. 2017 to Jul. 2018. At present, the dataset contains more than 501400 records that provide abundant data for our analysis.

B. Popularity Features and Potential Problems

By analyzing the dataset built, we conclude the 3 main popularity features of IoT data:

Regularity. IoT data's popularity changing has regularity, the popularity changes with basic period regularity but with many small saltation (see Fig. 2a). While in different regions and periods, the changing trend has its own characteristics. This mainly because the request of IoT data is highly related to regularity of people's life and needs (behaves as the spatial temporal relationship), and will be interfered by some accidents.

Zipf's law. We analyze the cumulative popularity of one day and find the popularity Zipf's law in the popularity index: $P_D = \frac{C_D}{D^{\alpha_d}}$, where P_D is the cumulative popularity of the district D, C_D, α_d are parameters of Zipf's law (see a case in Fig. 2b). However, we observe the the index sequence of popularity are not consistent but will change by different periods (see Fig. 2c), which indicates the spatio-temporal relationship of IoT data's popularity is time varying.

Timeliness. IoT data's popularity has timeliness characteristic due to the value is valid in a specific time period. After the valid period[4], the popularity will experience a significant reduction (see a case in Fig. 2d). Because very few people are interest in history data whose value is invalid. Specifically, this reduction behaves an exponential decay process: $p_t = C/E^t$, where C and E is the parameter of the exponential function and t is the timestamp (in valid period level).

These observed features indicate the potential problems while using classical traffic model in IoT: Due to the timeliness feature, the request model based on popularity of whole content space (such as IRM) will lose efficacy [3]. Besides, the regularity and time varying features indicate the popularity changing can not be demonstrated by a simple function (e.g., SNM). Therefore, we analysis the request process of IoT data deeply so as to design the appropriate modelling method, as introduced in next sections.

C. Popularity Distribution

In this section, we derive the popularity distribution of IoT data. For the first, we regard the popularity as the normalized request frequency. Then, the request process $\{N(t), t > 0\}$ of the IoT data of a specific type in a region is a non-homogeneous poisson process due to it satisfies the following conditions:

$$\begin{cases} \{N(t), t_j \le t < t_{j+1}\} \text{ has independent increment}, \\ P\{N(t+\Delta h) - N(\Delta h) = 1\} = \lambda(t)\Delta h + o(\Delta h), \\ P\{N(t+\Delta h) - N(\Delta h) > 2\} = o(\Delta h), \end{cases}$$

where $\lambda(t)$ is the intensity function and Δh is the time slot. However, determining $\lambda(t)$ is extremely difficult because it is non-linear and full of saltation. Therefore, we shift from solving $\lambda(t)$ to determine the distribution of λ_j in specific time t_j. We divide the whole request process into individual changing period T and further divide T into different time slot S_j, which denotes $[t_{(j-1)k}, t_{jk})$, k is the length of the data's valid period. Thus we could reexpression $\lambda(t)$ as follows: $\lambda(t) = \{\lambda(S_j)\}, j \in [1, |T|]$. As for each S_j, the probability of arriving n requests are as follows:

$$P_n = \frac{\Lambda(S_j)^n}{n!} e^{-\Lambda(S_j)}, \tag{1}$$

where $\Lambda(S_j)$ the cumulative intensity function during S_j:

$$\Lambda(S_j) = \int_{t_{(j-1)k}}^{t_{jk}} \lambda(t)\mathrm{d}t. \tag{2}$$

Then, we calculate the eigenfunction of this process:

$$\begin{aligned} E(e^{itS}) &= \sum_{t_{(j-1)k}}^{t_{jk}} e^{itS} \frac{\Lambda(S_j)^n}{n!} e^{-\Lambda(S_j)} \\ &= e^{\Lambda(S_j)(e^{it}-1)}, \end{aligned} \tag{3}$$

where S indicates the variable in each S_j and $i = \sqrt{-1}$. Base on the conclusion of Formula (3), we can get the following theorem:

Theorem 1: *The popularity distribution of an IoT data during its valid time can be approximately considered as the normal distribution:* $p \sim N(\mu, \sigma^2)$.

Proof: During the valid time S_j of a data, the request process can be treated as a homogeneous poisson process due to it further satisfies the stationary increment condition. Therefore, we can get the form of Formula (3) in this scenario as follows:

$$E(e^{itS}) = e^{\lambda_j t(e^{it}-1)}, \tag{4}$$

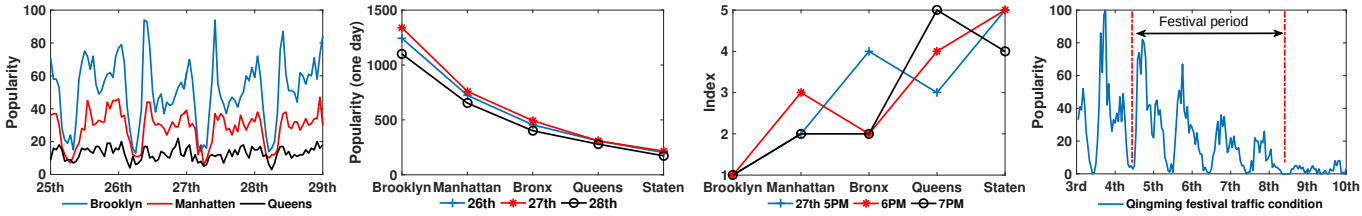
[2]https://trends.google.com/trends/

[3]https://index.baidu.com/

[4]In this paper, we use 1 hour as the valid period and 1 day as the changing period based on the publishing period of Beijing Air Quality Index (AQI).

TABLE I: Popularity Dataset of IoT data.

Dataset	City level		District level	
Locaion	Metropolises	Yangtze River Delta	Bejing	NYC
	London, Tokyo,...(5 cities)	Shanghai, Hangzhou...(12 cities)	Haidian, ...(10 districts)	Brooklyn, ...(5 districts)
Types	Temperature, light,...(5 types)	Humidity, PM2.5,...(7 types)	Weather (1 type)	Weather, traffic,...(5 types)
Time span	11/23/2017-7/2/2018	12/26/2017-7/24/2018	1/3/2018-4/10/2018	11/17/2017-7/2/2018
Source	Google	Baidu	Baidu	Google
Size	109200+	242300+	36500+	113400+
Overall records	501400+			



(a) Regularity (Nov. 2017) (b) Zipf's law (Nov. 2017) (c) Index changing (Nov. 2017) (d) Timeliness (Apr. 2018)

Fig. 2: Popularity features of IoT data.

where λ_j is the poisson intensity during S_j. We release time t to the time slot $\Delta t \in S_j$ and substitute $\rho = \lambda_j \Delta t$ into Formula (4):

$$e^{\rho(e^{it}-1)} = e^{\rho(it-\frac{t^2}{2!}-\frac{it^3}{3!}+\cdots)} \approx e^{\rho it-\rho\frac{t^2}{2!}},$$

where we omit the high order term $\frac{it^3}{3!}+\cdots$. Considering a variable $X \sim N(\rho,\rho)$:

$$\begin{aligned} E(e^{itX}) &= \frac{1}{\sqrt{2\pi\rho}}\int e^{itX-\frac{(X-\rho)^2}{2\rho}}dX \\ &= e^{\rho it-\rho\frac{t^2}{2}} \approx E(e^{itS}). \end{aligned} \tag{5}$$

Since the eigenfunction of S equals X, thus the distribution of S equals X [22]. If we normalize S into a fixed range (e.g., $S \in [0,100]$ in GT) as p, thus we have:

$$p \sim N(\rho',\rho'), \tag{6}$$

where ρ' the expected value and standard deviation after normalizing p. ■

Fig. 3 gives the fitting results of the popularity by normal distribution in different time scales. We note that the normal distribution may not obvious due to the randomness and saltation in larger/smaller period (could not maintain stationary increment). In the next section, we will introduce how to estimate ρ' and how to model the popularity changing.

V. Popularity Evolving Model

In this section, we introduce the Popularity Evolving Model (PEM). We first give the definition of PEM as follows:

Definition 1: A *Popularity Evolving Model (PEM), denoted by $\mathcal{M}$, is determined by four elements: $\mathcal{M} \triangleq \{R,H,F,\pi\}$, where R is the long term regularity, H is the near history trend, F is the feature space and π is the initiate state.*

Specifically, we assume T is the period length of a type of data in a specific region. As for every period, we use $\mathbb{E}_T(\rho'_i)$ to denote the expected parameter ρ'_i during S_i, which represents the regular popularity in T. Then, R can be illustrated as follows:

$$R_T = \{\mathbb{E}_T(\rho'_1), \mathbb{E}_T(\rho'_2), \cdots, \mathbb{E}_T(\rho'_T)\}, \tag{7}$$

Note that R_T will update with time to keep its accuracy in illustrating the regularity. Thus, we write this updating process as the state transition equation:

$$R_{T_{i-1}} \cdot E_T = R_{T_i}, \tag{8}$$

where E_T is the transfer matrix. As for H, we use p_t to denote the popularity at time t. Then, we use H_{t-1} to represent H from $t-T$ to $t-1$:

$$H_t = \{p_{t-T+1}, p_{t-T+2}, \cdots, p_t\}, \tag{9}$$

Similarly, we write the updating process of H as the state transition equation:

$$H_{t-1}K = H_t, \tag{10}$$

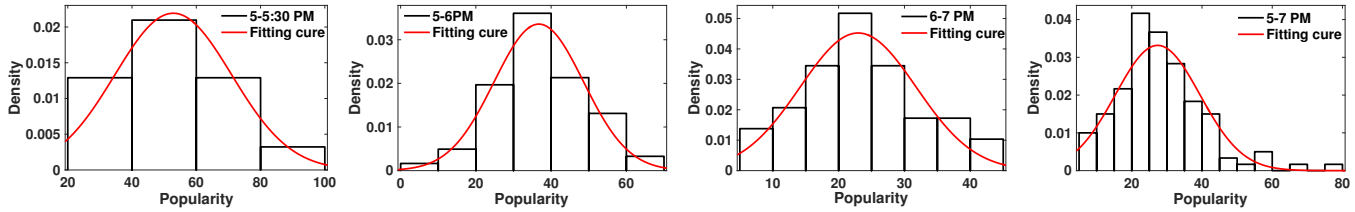
where E_T is the transfer matrix. PEM models the popularity changing of a specific type data in a region and indicates the evolving of the model. However, the key challenges are to determine the specific format of PEM and give the detail of evolving process. In next sections, we propose two methods: a simple but hight efficient modelling method called statistic based PEM and a complicated but high accurate data-driven method called DNN based PEM, to address this challenge.

A. Statistic Based PEM

In this part, we propose the Statistic Based PEM (SPEM): a simple but high efficient PEM modelling method. SPEM is constructed by following three steps: First, we rewrite Formula (8) as follows:

$$w_{T_{i-1}} R_{T_{i-1}} \cdot w_{T_i} E_{T_i} = R_{T_i}, \quad w_{T_{i-1}} + w_{T_i} = 1, \tag{11}$$

where $w_{T_{i-1}}$ and w_{T_i} are the weight to control the reserving and updating of long term regularity. The transfer matrix E_{T_i}



(a) Half hour (b) One hour (c) One hour (d) Two hour

Fig. 3: Normal distribution fitting in different time scales (27th, Nov. 2017 in NYC).

is constructed as a diagonal matrix that diagonal elements $p_{T_i t_j}$ denote the popularity of time t_j in T_i:

$$E_{T_i} = \begin{bmatrix} p_{T_i t_1} & & & \\ & p_{T_i t_2} & & \\ & & \dots & \\ & & & p_{T_i t_T} \end{bmatrix}_{T \times T}. \tag{12}$$

The transition process is significantly effected by the update weight $w_{T_{i-1}}$ and w_{T_i}. However, the history of previous R_T will be always kept when weight is chosen soever but cause different influence for now.

Second, we construct the transfer matrix of near history evolving as a time varying matrix as follows:

$$K = \begin{bmatrix} 0 & 0 & \cdots & 0 & k_1 \\ 1 & 0 & \cdots & 0 & k_2 \\ 0 & 1 & \cdots & 0 & k_3 \\ \vdots & \vdots & \vdots & 0 & \vdots \\ 0 & 0 & \cdots & 1 & k_T \end{bmatrix}_{T \times T}. \tag{13}$$

Using K in Formula (10) keeps the most of history information but only move a step forward by the last column, which is used to predict the popularity of next timestamp $\hat{p}_t$. Thus, we can get the estimation equation as follows:

$$\sum_{i=1}^{T} k_i p_{t-T+i-1} = \hat{p}_t. \tag{14}$$

This equation can be solved by the following theorem:

Theorem 2: *As for an IoT data, its popularity $\hat{p}_t$ at time t can be estimated as follows:*

$$\hat{p}_t = \mathbb{E}\left(p_{t-T}\right)\left(1 + \sum_{i}^{T-1} W_{t-T+i} \frac{\sigma^2(p_{t-T+i})}{p_{t-T+i}}\right), \tag{15}$$

where $\sigma^2(p_{t-T+i})$ is the estimated deviation and W_{t-T+i} is the weight.

Proof: According to our analysis in Sec. IV-B, $\hat{p}_t$ follows the long term regularity and behaves the normal distribution with some deviation influenced by near history:

$$\hat{p}_t = \mathbb{E}\left(\hat{p}_t\right) + \sigma^2(\hat{p}_t), \tag{16}$$

where $\mathbb{E}\left(\hat{p}_t\right)$ is the expected popularity estimated by long term regularity, and $\sigma^2(\hat{p}_t)$ is the estimated deviation. We divide Formula (16) into two parts: $\mathbb{E}\left(\hat{p}_t\right)$, which could be treated as a mapping from long term regularity, and $\sigma^2(\hat{p}_t)$, which could be treated as a mapping from the near history trends, i.e.,

$$\mathbb{E}\left(\hat{p}_t\right) = \mathbb{E}\left(p_{t-T}\right), \tag{17}$$

$$\sigma^2(\hat{p}_t) = \mathbb{E}(\hat{p}_t) \sum_{i}^{T-1} W_{t-T+i} \frac{\sigma^2(p_{t-T+i})}{p_{t-T+i}}. \tag{18}$$

Then, substituting Formula (17) and (18) into Formula (16), we can obtain Formula (15). ■

Finally, we construct the decay process of p_t to represent the timeliness of the data generated at time t. Hence, E_{T_i} is decaying as follows:

$$E_{T_i^j} = E_{T_i} D^{j-i}, j \in (i, +\infty), \tag{19}$$

where D is the diagonal matrix with diagonal element $e^{-\lambda}$, j is the timestamp and λ is the decay parameter. Now, a complete SPEM is constructed.

Constructing SPEM only needs small dataset and low computing resource, thus it has great advantage in deal with cold boot problem and can adapt for resource-constrained environments. Moreover, we note that even the weight is chosen by the simple statistic and empirical method, SPEM could performance good in modelling results (see Sec. VII).

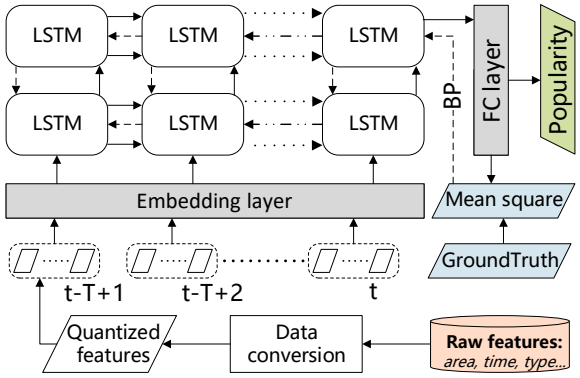


Fig. 4: DNN architecture.

B. DNN Based PEM

In this part, we propose the DNN based PEM to generate the high accurate popularity model. Because of DNN's advantages in automatical parameter learning/training and feature abstracting, we think DNN is a promising way to construct PEM and model the popularity changing process. However, the key challenges are: how to design a appropriate DNN architecture, and how to provide enough robust features for DNN training.

To solve the first challenge, we design a DNN architecture named Embedding Multi-layer LSTM (EML) (see Fig. 4). Specifically, we use LSTM as the core of EML. As a special neural network of Recurrent Neural Network (RNN), LSTM is widely used in Natural Language Processing (NLP) fields. LSTM overcomes the gradient vanishing problem by its special gate mechanism such that it can capture the long term memory of history data due to the memorying/forgeting process. Therefore, LSTM could capture the long term regularity and near history trend of IoT data's popularity, and could automatical adjust the weight from large history dataset by back propagation.

In this paper, we use two-layer LSTM (with 128 hidden cells) instead of traditional single-layer to improve the training efficiency and accuracy [23]. As for every timestamp, the updating details of LSTM are as follows:

$$\begin{cases} f_t = & \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\ i_t = & \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\ \hat{C}_t = & tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \\ C_t = & f_t * C_{t-1} + i_t * \hat{C}_t, \\ o_t = & \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\ h_t = & o_t * tanh(C_t), \end{cases}$$

where W means the weight and b means the offset deviation. The output of LSTM in layer 1 will be used as the input of LSTM layer 2. And the output of LSTM layer 2 will be input to a full connection (FC) layer with a linear activation function. The final output will be used to calculate the loss of N mini-batch by compared with ground truth, and back propagate (BP) to adjust the weights: $Loss = \frac{1}{N}\sum_i^N |p_{i\_estimated} - p_{i\_truth}|$.

To deal with the second challenge, we use the high-dimensional space embedded method to provide more robust features for DNN training. As a data driven model, DNN requires large dataset and enough features for network training. However, the features of IoT data are much little compared with images and videos that usually used in DNN. Besides, when we do not have enough data, a little saltation may greatly influence the modelling process and lead to the overfitting problem.

Our solution is as follows: First, based on the analysis of popularity changing in Sec. IV-B, which indicates the main features of IoT data, we use the spatio-temporal information combing with data types as the sematic features. These features are combined with the popularity of previous timestamp and area where the request is made as the 5 dimension raw features. Then, we quantize these raw features as DNN format vectors and use a embedding layer to converse them to the high dimension space: $f(\mathbb{R}^n) \mapsto \mathbb{R}^m$, $n \leq m$. Such embedding techniques are proposed to solve the Word to Vector (W2V) problem in NLP such as dimensionality reduction [24]. In this paper, we use embedding layer for dimensionality promotion to improve the robust of IoT data's features. Moreover, we find the promotion level shows good performance in the middle size of features' value domain:

$$a = \mathbf{F} : \{t_i, D_i ...\}, \tag{20}$$

$$f(a) \mapsto V :< v_1, v_2, ...v_{\frac{k}{2}} >, \quad a \in (1, k), \tag{21}$$

where $\mathbf{F}$ is the feature set the IoT data and V is the mapping value which follows the normal distribution:

$$V \sim \mathbf{N}(\mu, \sigma^2), \quad V \in \mathbb{R}^m. \tag{22}$$

Finally, we feed the features to LSTM core by using one period data as a set, which provides the near history trends.

In the training process, EML uses *Adam* algorithm [23] as the optimizer in the network and initiate the learning rate as 0.0005. The training epoch is 1000 where batch size equals 24×3. Note that DNN learning core runs offline thus do not consume the computing resources in local cache module, in where only need to fetch the generated model with little cost in communication.

VI. Popularity-based Caching Alghorithms

In this section, we first propose the popularity based cache algorithms including evicting and prefetching algorithm. Then, we give the theory analysis of the caching performance.

Algorithm 1 Popularity based evicting algorithm.

Input: Current arriving data D_c; Cached data sequence $\{D_i\}$ in ascending order of popularity; Generated popularity model $\mathbb{P}$;

Output: Awaiting evicting data D_e;

```
Check the cachable characteristics of D_c;
if cacheable! = True then abandon D_c; return NULL;
end if
if cache!=Full then cache D_c; return NULL;
end if
Find last item D_last in {D_i} and do D_e ← D_last;
Calculate popularity p_c ← (D_c ⇒ ℙ);
Insert D_c → {D_i};
Re-sorting the sequence {D_i} according to {p_i};
return D_e;
```

A. Evicting and Prefetching Algorithm

In this part, we detail the evicting and prefetching algorithm based on the popularity model generated before. By using the model to predict the IoT data's popularity, the evicting algorithm makes the decision of "what to cache" and the prefetching algorithm makes the decision of "when to cache".

Algorithm 1 illustrates the popularity based evicting algorithm. For the first, the algorithm sets a step to verify the *cacheable* characteristic of the data to filter the "un-cacheable" data such as: control commands, error reports, etc. Then, the passed data will be cached if the cache is not full. Otherwise, the algorithm evicts the data with the lowest popularity in the previous data sequence. After that, the algorithm uses the popularity model to predict the popularity of the incoming data and the data will be cached within the new data sequence re-sorted by their popularity.

Algorithm 2 illustrates the popularity based prefetching algorithm. For the first, the algorithm calculates how many data can be prefetched according to the size of cache. Then, it generates a cumulative function of the popularity and set the threshold according to the cache size. As for each period, the algorithm uses the popularity model to predict the popularity of the data space and prefetches the data whose popularity is higher than the threshold.

B. Analysis of Cache Performance

In this part, we analyze the cache performance while using the proposed cache algorithms. For the first, we normalize the $\{P_{V_i}\}$ into the domain $(0, 1)$ as $\{\hat{P}_{V_i}\}$, which represents the

Algorithm 2 Popularity based prefetching algorithm

Input: Current period T_C;
Output: Awaiting prefetching data sequence $\{D_p\}$;

1: **Initialized notation:** S is the cache space allocated for prefetching, V_p is the data space of current period. F_{V_p} is the cumulative popularity function of current period. Define the $\zeta = \frac{S}{V_P}$ as the fraction of the data that could be cached in cache space, φ as the top $\zeta - th$ in the percentile of F_{V_p}. Then we can get: $\varphi = F_{V_P}^{-1}(1-\zeta)$.
2: **Procedure begin:**
3: Trigger procedure at the beginning of T_C.
4: Construct the V_p for popularity model $\mathbb{P}$, return predicted popularity sequence $\{P_{V_i}\}$.
5: **for** P_{V_i} **in** $\{P_{V_i}\}$ **do**
6: **if** $P_{V_i} > \varphi$ **then** $D_{V_i} \rightarrow \{D_p\}$;
7: **end if**
8: **end for**
9: **return** $\{D_p\}$;

request probability of the item in V_p. Then, we can get the following theorem:

Theorem 3: *The hit ratio of a node using popularity based evicting and prefetching algorithms is limited in the range as follows:*

$$\sum_{j=V_P-\varphi}^{V_P} \hat{P}_{V_j} + \sum_{i=1}^{\varphi} \hat{P}_{V_i} \leq H \leq \sum_{i=1}^{S} \hat{P}_{V_i}. \tag{23}$$

Proof: The cache node using Algorithm 2 will prefetch the data ranges from 1th to φth in F_{V_p}. Therefore, we could get the rough hit ratio at the beginning: $H = \sum_{i=1}^{\varphi} \hat{P}_{V_i}$. Then, Algorithm 1 ensures these data will be hold them in the storage cache. Once the cached data full of the cache, the worst phenomenon is that the node caches the data with the lowest popularity ranges from $(V_P - \varphi)$-th to V_P-th. In this scenario, the lower bound is $\sum_{j=V_P-\varphi}^{V_P} \hat{P}_{V_j} + \sum_{i=1}^{\varphi} \hat{P}_{V_i}$. In contrast, the best phenomenon is that the node caches the data with the highest part that ranges from 1st to Sth. Therefore, the upper bound is $\sum_{i=1}^{S} \hat{P}_{V_i}$. ■

From Theorem 3, the proposed algorithms address the "first missing" problem, i.e., the data requested for the first time can not be hit in the cache. Even at the beginning, the node can get relative high hit ratio due to the prefetched data with high popularity in the cache.

Moreover, if we know the precise distribution of popularity, we can calculate H accurately. For instance, assuming the popularity follows a simple uniform distribution, we can get the H as follows:

$$H = \sum_{i=1}^{\varphi} \hat{P}_{V_i} + \frac{S-\varphi}{V_p}. \tag{24}$$

In the next, we will give the evaluation of IoTCache.

VII. Evaluation

A. Experiment settings

Experiment platforms. In order to evaluate the proposed IoTCache, we develop two platforms: CDN-IoT (CoT, for short) and NDN-IoT (NoT, for short). As for CoT, we develop a typical CDN architecture [5] and deploy it in a star network with 5 nodes (i7-4790 3.6GHz CPU, 16G memory) as the IoT backbone. Specifically, we use 1 node as the content distribution center to interconnect content provider i.e., edge sensors[5] and 4 nodes as the edge server that provide content service to users. As for NoT, we ultilize NFD [25] to deploy NDN as the IoT backbone. Specifically, NoT consists of 1 center router, and 4 edge routers. We modify the edge routers (by modifying NFD) as the IoT gateways so that could provide the interconnection between edge sensors and NoT backbone. For all nodes in CoT/NoT backbone, we deploy typical two-layer cache [26]: Hot Object Cache (HOC) using RAM/ROM as the layer 1 (CS in NDN) and Disk Cache (DC) as layer 2 (Repo in NDN). When the request can not be satisfied within the backbone, it will be delivered to edge sensors.

TABLE II: MAE on 4 datasets.

Location	NYC	Beijing	Metropolises	Yangtze	Average
LR	12.5	22.2	10.4	20.1	16.3
SPEM	9.97	13.3	8.09	11.15	10.6
LSTM	7.33	11.07	5.64	12.4	9.11
EML	**7.11**	**6.35**	**4.83**	**7.61**	**6.48**

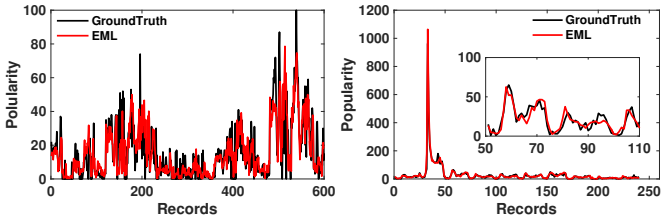
Evaluation metrics. The evaluation is divided into two parts: modelling performance and cache performance. For the first, we compare the modelling accuracy of our proposed SPEM and EML with LR and LSTM. We implement SPEM by Python and use the weights, deviation and decay parameters as follows: $w_{T_{i-1}} = w_{T_i} = 2^{-1}, W_i = 2^{-i}, \sigma^2(p_i) = p_i - \mathbb{E}(p_i), \lambda = 10$. The modelling accuracy is evaluated by mean absolute error (MAE): $MAE = \frac{1}{N}\sum_{i=1}^{N} |(p_i - p_{mi})|$, where p_{mi} and p_i are the modelling value and ground truth, respectively. N is the number of all modelling values. Moreover, we compared the computational time (CT) of each method executed in a same computer (Intel i7-4790 3.6GHz CPU and 16G memory without GPU). The developing platform of all neural network (LR, LSTM, EML) is TensorFlow, in which we use data of one month (80% as train set, 20% for validation set) to train the model and five days as the test set.

Secondly, we compare IoTCache with FIFO and LRU in NoT and CoT respectively. The evaluation metrics are: cache hit ratio, edge packet ratio and data latency. We develop a discrete event simulator to generate the request (Interest in NDN). The request process follows the conclusion in Sec. IV-C and the popularity of the each period is directly obtained from the dataset built. The test period is the same day as used in comparing modelling accuracy. All types of the experiments are repeated for 5 times and we use the average value of all nodes to reduce the random error.

B. Experiment Results

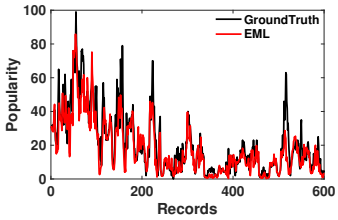
Modelling error. Table II illustrates the MAE value of compared methods and our proposed EML. As shown in the result, LR shows a bad performance in all dataset (average MAE: 16.3) due to its weak ability in modelling such complex system, especially for saltation during the change. SPEM reduces the MAE by 20.2%, 40.1%, 22.2%, 44.5% for NYC, Beijing, Metropolises and Yangtze River Delta dataset, respectively and 35% for average. These results show that SPEM outperforms the LR in illustrating the IoT popularity changing.

[5]We use the TelosB node of CrossBow Inc. as the edge sensor.

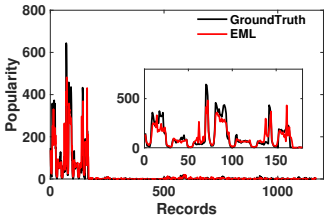


(a) NYC.

(b) Beijing.

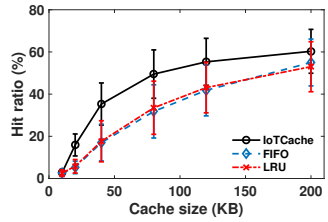


(c) Metropolises.

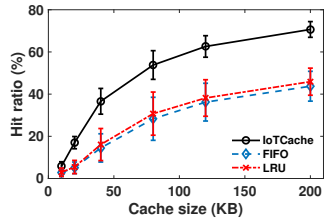


(d) Yangtze River Delta.

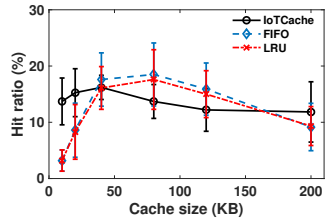
Fig. 5: Experiment results of modelling accuracy on 4 datasets.



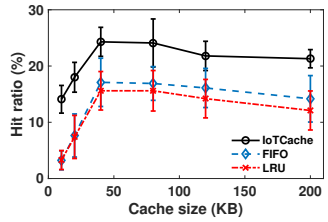
(a) Hit ratio in CoT HOC.



(b) Hit ratio in NoT HOC.



(c) Hit ratio in CoT DC.



(d) Hit ratio in NoT DC.

Fig. 6: Experiment results of the hit ratio.

Compared with SPEM, LSTM reduces the average MAE by 14%, which indicates the good performance of DNN based training methods. EML, which has the lowest MAE among the all methods, reduces the average MAE by 60.2%, 38.9%, and 28.9% compared with LR, SPEM, and LSTM, respectively. Fig. 5 illustrates the modelling results and ground truth, which shows a great matching in different dataset. Therefore, we could claim that out proposed EML can accurately illustrate the popularity changing in IoT.

Computational time. Table. III gives the computational time of all compared regression methods. The called function is *time.clock()*, which only considers the CPU running time in modelling one day's popularity. As shown in the results, LR consumes the lowest computational time (the only one in *ms* level) due to its simple modelling process while also bring with the worst modelling accuracy. As for SPEM, LSTM, and EML, the average computational time are $2.8ms$, $0.06s$, and $0.18s$ respectively. EML consumes the highest time among four modelling methods. However, this processing time is low enough for using in most of IoT environments.

TABLE III: CT on 4 datasets.

Location	NYC	Beijing	Metropolises	Yangtze	Average
LR	0.71ms	0.69ms	0.5ms	0.4ms	0.58ms
SPEM	2.6ms	1.1ms	2.6mss	4.8ms	2.8ms
LSTM	0.05s	0.02s	0.06s	0.11s	0.06s
EML	**0.13s**	**0.04s**	**0.12s**	**0.26s**	**0.18s**

Hit ratio. In this part, we compare all involved caching algorithms with cache hit ratio in HOC and DC respectively. The cache size ranges from 10KB to 200KB and keep the same in HOC and DC. Fig. 6 illustrates the experiment results of cache hit ratio. As we can see from the results, LRU and FIFO always keep the same trends in both CoT and NoT. This mainly because they are both based on the temporal locality of the request process to evict the content. While in the environment with sparse requests, there will be no significant difference between them.

More important, the results show that IoTCache keeps higher hit ratio than compared algorithms in both CoT and NoT. In CoT, when cache size of HOC equals $40KB$, IoTCache outperforms the LRU/FIFO about **2**$\times$ (35.3% to 17%). When cache size of DC equals $10KB$, IoTCache outperforms FIFO/LRU up to **4.3**$\times$ (13.7% to 3.2%). However, when cache size ranges from $50KB$ to $150KB$, we can see the hit ratio of IoTCache in DC is lower than others. This mainly because one node using IoTCache reaches a high hit ratio in HOC ($>$82%) and reduces the average hit ratio of DC during this range. In NoT, IoTCache also outperforms others in both HOC and DC in every cache size level. Finally, when cache size equals $200KB$, IoTCache improves the hit ratio by 25% and 7% in HOC and DC respectively.

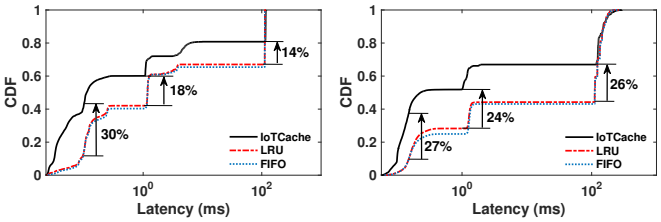
The main reason is that LRU and FIFO are difficult to track the fast variations of popularity by using the local and small history request traces. In contrast, IoTCache is based on the accurate popularity prediction via the popularity model. Therefore, IoTCache can make precise evicting/prefetching decision to store the high popularity data. This advantage is very useful when we deploy the IoTCache in storage-constrained IoT environment with sparse requests.

Edge packet ratio. This part gives the evaluation of the edge packet ratio (EPR). Specifically, EAR is calculated as follows: P_{edge}/P_{all}, where P_{all} is the number of all packets and P_{edge} is the number of packets in edge networks. By using EPR we can ignore the specific number of the packets in CoT/NoT but to focus on the transmission ratio in IoT edge. In this experiments, cache size is $100KB$. Table. IV illustrates the experiment results: In CoT, the EPR of IoTCache, LRU, FIFO are 15%, 24% and 25% respectively. In NoT, these values are 7.2%, 33% and 32% respectively. Compared with LRU and FIFO, IoTCache can decrease EPR about 10% (in CoT) and 26% (in NoT). This is because the high hit ratio and accurate prefetching mechanism of IoTCache can decrease the repeat transmission in IoT edge, and also gain great reduction of traffic in backbone. Moreover, because all nodes in NDN are cacheable, thus NoT can further decrease the traffic in IoT edge while the request has more chance to be satisfied within the backbone of NDN.

Data latency. In this experiment, we set the same cache size ($100KB$) and compare the data latency of IoTCache with others. The latency is defined as the time interval between

TABLE IV: Edge packets ratio.

Platform	CoT			NoT		
Strategy	IoTCache	LRU	FIFO	IoTCache	LRU	FIFO
EPR	15%	24 %	25%	7.2%	33%	32%



(a) Latency in CoT (b) Latency in NoT

Fig. 7: Latency in CoT and NoT.

sending request and receiving data. We plot the results as Cumulative Distribution Function (CDF) thus could compare the distribution details and the ratio while in different latency scales. As shown in Fig. 7a, CDF values of IoTCache are always higher than LRU/FIFO. Specifically, when latency level is $0.1ms$, CDF values are 45.7%, 15.5% 14.8% for IoTCache, LRU and FIFO respectively. IoTCache improves the CDF value about 30%. When latency levels are $1ms$ and $100ms$, the improved values are 18% and 14% respectively. The main reason is that the high cache hit ratio in cache can shorten the fetching delay especially when hit in HOC cache. While in CoT, the high hit ratio in edge CDN servers can further shorten the delay between the center and edges. Similar phenomenon also happens in NoT, as shown in Fig. 7b, when latency levels are $0.15ms$, $1ms$ and $100ms$, we can see that IoTCache improves the CDF value of latency by 27%, 24%, and 26%, respectively. Moreover, we find that the overall latency of NoT is higher than CoT, which is mainly due to the relative higher hops of NoT (from one edge to others) compared with CoT (from the edge to the center).

VIII. Conclusion

In this paper, we propose IoTCache, a popularity based caching solution for IoT. Our contributions are three folds: Firstly, we build a large popularity dataset. Base on the analysis of the dataset, we derive the popularity distribution and further propose PEM to model the popularity changing. Secondly, we propose the data-driven popularity modelling method that can accurately predict the future popularity of IoT data. Thirdly, we design the popularity-based evicting and prefetching algorithms. We implement IoTCache in developed IoT platforms based on CDN and NDN respectively. The evaluation results in these platform show that IoTCache gains a significant improvement in cache hit ratio and reduction in edge traffic and data latency compared with traditional methods. IoTCache still has some weakness. For example, DNN learning core requires large dataset and high performance computer for training, which may difficult for resource-constrained users. Our future work will mainly focus on deploying IoTCache in COTs devices used in IoT, and explore the other application scenarios of IoTCache.

References

[1] J. Gubbi et al. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[2] H. Ma, L. Liu, A. Zhou, and D. Zhao. On networking of internet of things: Explorations and challenges. *IEEE Internet of Things Journal*, 3(4):441–452, 2016.

[3] Georgios Paschos, Ejder Bastug, Ingmar Land, Giuseppe Caire, and Mérouane Debbah. Wireless caching: Technical misconceptions and business barriers. *IEEE Communications Magazine*, 54(8):16–22, 2016.

[4] George Pallis and Athena Vakali. Insight and perspectives for content delivery networks. *Communications of the ACM*, 49(1):101–106, 2006.

[5] Jaeyoung Choi, Jinyoung Han, Eunsang Cho, Ted Kwon, and Yanghee Choi. A survey on content-oriented networking for efficient content delivery. *IEEE Communications Magazine*, 49(3), 2011.

[6] B. Ahlgren et al. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7), 2012.

[7] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017.

[8] Evangelos K Markakis, Kimon Karras, Anargyros Sideris, George Alexiou, and Evangelos Pallis. Computing, caching, and communication at the edge: The cornerstone for building a versatile 5g ecosystem. *IEEE Communications Magazine*, 55(11):152–157, 2017.

[9] Xiaofei Wang, Min Chen, Tarik Taleb, Adlen Ksentini, and Victor Leung. Cache in the air: exploiting content caching and delivery techniques for 5g systems. *IEEE Communications Magazine*, 52(2):131–139, 2014.

[10] Akamai. Over the air (ota). https://www.akamai.com.

[11] Microsoft. Microsoft azure cdn. https://azure.microsoft.com/en-us/services/cdn/.

[12] Serdar Vural, Pirabakaran Navaratnam, Ning Wang, Chonggang Wang, Lijun Dong, and Rahim Tafazolli. In-network caching of internet-of-things data. In *Communications (ICC), 2014 IEEE International Conference on*, pages 3185–3190. IEEE, 2014.

[13] Dong Doan Van and Qingsong Ai. An efficient in-network caching decision algorithm for internet of things. *International Journal of Communication Systems*, 31(8):e3521, 2018.

[14] Bo Chen, Liang Liu, Zhao Zhang, Wenbo Yang, and Huadong Ma. Brr-cvr: A collaborative caching strategy for information-centric wireless sensor networks. In *Mobile Ad-Hoc and Sensor Networks (MSN), 2016 12th International Conference on*, pages 31–38. IEEE, 2016.

[15] Suoheng Li, Jie Xu, Mihaela Van Der Schaar, and Weiping Li. Popularity-driven content caching. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[16] Edward Grady Coffman and Peter J Denning. *Operating systems theory*, volume 973. Prentice-Hall Englewood Cliffs, NJ, 1973.

[17] Stefano Traverso, Mohamed Ahmed, Michele Garetto, Paolo Giaccone, Emilio Leonardi, and Saverio Niccolini. Temporal locality in today's content caching: why it matters and how to model it. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, 2013.

[18] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[21] M. Leconte et al. Placing dynamic content in caches with small population. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[22] Joseph L Doob and Joseph L Doob. *Stochastic processes*, volume 7. Wiley New York, 1953.

[23] J. Donahue et al. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[24] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[25] A. Alexander et al. Nfd developers guide. 2015.

[26] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *NSDI*, pages 483–498, 2017.