Received 15 July 2024; revised 11 August 2024; accepted 18 August 2024. Date of publication 23 August 2024; date of current version 5 September 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3449241

Predictive Caching in Non-Stationary Environments: A Time Series Prediction and Survival Analysis Approach

JAVANE ROSTAMPOOR [1] (Member, IEEE), RAVIRAJ S. ADVE [1] (Fellow, IEEE), ALI AFANA [2], AND YAHIA A. ELDEMERDASH AHMED [2]

[1]Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 1A1, Canada

[2]Ericsson Canada, Ottawa, ON K2K 2V6, Canada

CORRESPONDING AUTHOR: J. ROSTAMPOOR (e-mail: jrostampoor@ece.utoronto.ca)

This work was supported by the Ericsson Canada and by the National Sciences and Engineering Research Council (NSERC) Canada.

ABSTRACT This paper introduces an innovative predictive caching strategy tailored to a real-world dataset, specifically the Facebook video dataset. Making caching decisions for the dataset is challenging due to its dynamic nature, where users' content requests vary over time without fitting into any known models. Traditional caching strategies, which often rely on a constant pool of files, do not suit this dataset as content is requested by users, and then its popularity fades over time; furthermore, the list of available content changes. We propose a two-stage predictive caching strategy. Initially, it forecasts the number of user requests using content features and historical request data, achieved through training a long short-term memory (LSTM) network. Then, we employ our proposed extended Cox proportional hazard (E-CPH) model to predict the survival probability of content. This facilitates proactive content caching. Caching new content is made possible by the timely eviction of content unlikely to be requested again. To incorporate the predicted content popularity and its life cycle into the caching decision, we introduce a partially observable Markov decision process (POMDP)-based caching strategy. Here, the survival probability of content contributes to the belief state of the associated content which leads to our believed predicted reward - a cache hit. The caching algorithm then stores the files based on their predicted believed reward taking into account both the popularity and survival probability predictions. Simulation results validate the efficacy of our proposed predictive caching method in enhancing the cache hit rate compared to conventional recurrent neural network (RNN)-based caching and policy-based caching approaches, such as least frequently used caching and its variants.

INDEX TERMS Predictive caching, LSTM networks, Cox proportional hazard model, POMDP, cache hit maximization.

I. INTRODUCTION

THE RAPID development of the 5th generation (5G) of wireless networks and the Internet of things (IoT) has led to a remarkable increase in network traffic [1]. In addition, with the continuing implementation of 5G networks, research and development are focused on 6th generation (6G) wireless networks wherein the increase in traffic volumes is expected to continue [2]. Our emphasis lies on caching, which brings content closer to the network edge, thereby reducing strain on both fronthaul and backhaul links. This approach also allows for quicker response times to user requests, addressing them with minimal latency [3].

Caching strategies can be reactive or proactive [4]. Reactive caching determines whether to cache a file upon considering requests; caching decisions are based on factors like its popularity, available cache space, etc. In contrast, proactive caching forecasts future requests for files by analyzing their request history and features such as the genre of films, length of file, etc.

Due to the dynamic nature of content requests, popularity prediction is a key enabler of effective proactive caching [5].

The first crucial step in video popularity prediction is gathering data information, known as features, which include static, temporal, social, and cross-domain features [5]. Static features are fixed attributes prepared before the video's publication, such as category, video quality, title, and description. Temporal features refer to dynamic aspects such as the number of views for each video, the age of the video, and so on. Social interactions, which can significantly influence the future popularity of the content, are modeled by social graphs, where users are represented as nodes and the relationships among them as edges. Understanding social interactions in advance can aid in predicting future content requests. Cross-domain features are extracted from external sources and can be instrumental in forecasting future content demand. For example, features from Twitter can be utilized in prediction models to determine the popularity of YouTube videos [6].

Predicting future trends by leveraging historical data on content requests, which assumes a correlation between past and future requests, has been studied in the literature [7], [8]. The authors believe that historical information about data is embedded in past content requests, as there is a correlation between past and future requests. In this approach, the past video popularity data is utilized to forecast what comes next. A similar approach is used in [9], [10] to predict future workloads and optimize resource utilization in the network. The introduction of a segment-based prediction and cache replacement strategy in [11], [12] represents an improvement, despite its greater computational complexity. This strategy, using the proximity of segments as a predictive tool, leverages the relationship between video segments to make more precise predictions about the popularity of future segments. To reduce computational complexity, [13] proposes an autoregressive (AR) model. This model utilizes sufficient durations of time series data for accurate prediction, while also aiming to reduce network costs and improve cache hits in the network (cache hits are defined as the summation of the fraction of total requests directly serviced by the cache, where there is no need to fetch data to respond to the users' requests). It is worth noting that these methods all assume that requests follow Zipf's distribution.

The authors in [14] move towards online prediction, adapting a popularity model to accommodate variations in request patterns. They define a smoothing parameter that quantifies the influence of past data on future predictions. For new videos for which enough historical data is unavailable, video metadata and user data such as watch time, shares, subscribers, and video age can be considered. In [15], a hybrid approach to predict content popularity, combining offline and online methods is introduced, based on multivariate regression analysis on video features. This approach starts with fixed coefficients derived from offline training and dynamically updates them using an exponential weighted moving average (MA), effectively addressing the limitations in both offline and online models.

Popularity-based caching has been studied in [16], [17]. In [16], the authors address popularity-based caching in cloud radio access networks (C-RANs) by formulating an optimization problem to maximize effective capacity. This approach combines predicting content popularity and user mobility through a deep learning framework. Predictive methods include estimating the likelihood of content requests and forecasting user locations, facilitating optimal caching decisions in the cloud, etc. Meanwhile, [17] introduces the use of social graphs to improve content request predictions. These graphs help predict video popularity spread and future view increases, with content retention in caches adjusted accordingly. However, the dynamic nature of social interactions and the extensive dependent information they generate can complicate network management.

In this paper, the ultimate goal is to analyze real-world traffic, make predictions and use the information to make caching decisions for a limited-capacity caching network. The caching optimization can be implemented at the base station, where user request information can be effectively gathered and utilized. However, the location of the cache is flexible and can be adjusted according to the network topology. We make caching decisions to maximize cache hit rate, which is the summation of the total requests that are directly addressed by the cache, eliminating the need to fetch data to respond to the users' requests.

It is evident that storing the most popular files would lead to cache hit maximization. However, due to the environment's dynamics, the content we consider popular now might not be popular in the future or even might be eliminated. In this regard, instead of modeling the environment, we interact with the environment to predict user requests and make caching decisions. Our predictive caching strategy includes two stages to learn the environment: predicting the number of user requests based on time series forecasting methods and predicting the requests' popular time or life cycle. Subsequently, by integrating this knowledge into a partially observable Markov decision process (POMDP), we aim to maximize cache hits within the network.

Our prediction strategy is based on time series forecasting of user requests assuming that the requests will survive for the upcoming time slots. To make the pattern predictions, we use a long short-term memory (LSTM) network to learn the dataset. Then, it is the responsibility of a survival time analyzer to predict the probability of whether the content survives or dies. We use an extended Cox proportional hazard (E-CPH) survival model to find the survival probabilities of different content in the future. In developing E-CPH, we extend the original CPH approach using a wiser choice of inputs.

At each time block, we need to make caching decisions based on two sets of information: obtained content popularity evolution and survival probabilities. We model sequential decision-making as a POMDP in which the state of the

content is not fully observable. In other words, we have predicted content popularities for the future; however, we are not sure whether the state of the content is alive or dead. Survival probabilities introduce extra information as beliefs about states which helps us to make caching decisions based on the believed reward, which is the number of cache hits here.

The main contributions of this manuscript can be summarized as:

- Providing a predictive caching solution for a real-world dataset without common assumptions in the literature, such as assuming the user requests are drawn from a known popularity distribution or having a known or unlimited life cycle.
- Introducing a time series prediction approach based on LSTM networks with a custom loss function that highlights our caching goal.
- Taking into account the limited life cycle of user requests and enhancing the available features to improve learning the life cycle with our introduced E-CPH method.
- Even though the environment does not follow Markovian evolution, with the aid of life cycle predictions, we inject the information of the life cycle as extrinsic information on the beliefs to approximate our optimization problem with an MDP.

The rest of this paper is organized as follows: Section II introduces the related works. Section III explains the characteristics of the required dataset for caching decisions and introduces the dataset used along with its features and details the pre-processing steps to prepare the dataset for further processing. In Section IV, we propose our popularity forecasting model, featuring our LSTM predictor network. Section V introduces our E-CPH survival model, and Section VI develops a POMDP-based caching strategy that combines request prediction outputs with survival probabilities. Section VII presents numerical results to validate the effectiveness of our approach in a real setting and compares its performance with previous caching methods. Finally, Section VIII concludes the paper.

II. RELATED WORKS

Our work lies at the intersection of prediction-based caching and caching under uncertain conditions. An LSTM network, a special type of recurrent neural network (RNN), is renowned for its ability to learn and remember long-term dependencies [18]. In this regard, LSTM networks are suited for predicting patterns in time series data, a capability crucial for forecasting requests in caching scenarios [19], [20], [21], [22], [23], [24]. The authors in [19] propose LSTM-C, an edge-assisted caching replacement algorithm that learns cache replacement strategies from past request sequences without needing data pre-processing or additional information. The study in [20] focuses on cache hit maximization based on proactive caching using an LSTM network. First, each user group is trained on an LSTM model; the results are then averaged to obtain the average demand for user groups to generate an efficient caching policy. The authors in [21] employ an LSTM architecture to make user popularity requests and trajectory predictions to cache popular content in the best cache locations. Similarly, in [22], a deep learning architecture named Stacked Autoencoder-Long Short Term Memory Network (SAE-LSTMNet) is proposed to capture both the correlation of request patterns among different content files and the periodicity in the time domain to improve prediction accuracy of content popularity. In [23], a user preference learning-based proactive edge caching (UPL-PEC) strategy is developed, utilizing bidirectional LSTMs and graph convolutional networks to predict user preferences and optimize caching strategies. Finally, the study in [24] presents a clustering-based LSTM (C-LSTM) approach to predict content requests for optimizing caching policies in wireless coded caching.

The impact of uncertainty on caching decisions has also been studied in the literature [25], [26], [27]. In [25], the authors describe the uncertainty in predicting popularity trends by using a Gaussian distribution to represent the additive errors in predictions. In [26], the approach is slightly different, with the added errors being represented by a uniform distribution. The authors also look at files' ranking errors within a set library, assuming these rankings are based on predicted popularity for caching purposes. However, the models used in these studies are based on assumptions on popularity distributions and may not accurately capture the changing nature of what becomes popular. In [27], the authors address the uncertainties associated with dynamic popularities by considering additive errors in predictions, which arise from both prediction and estimation inaccuracies, miss alarms, which are mainly caused by the arrival of new popular files that were not anticipated, and false alarms, which occur primarily due to the aging of files that were previously popular.

In this paper, we analyze real-world user request data, which shows dynamic content requests over time with probability of the content vanishing. Hence, in our dataset, due to the environment's dynamics, what we now consider popular might not be popular in the future or even might be dead. This study is different from the introduced proactive caching studies. Unlike [19], [20], [21], which overlook the finite nature of user request lifecycles, or [27], which assumes a long lifespan for the user requests, or even [28], which considers a known distribution for the user requests' lifecycle, we explicitly acknowledge the variable and limited duration of content requests over time.

In addition, instead of modeling the environment and making decisions based on the model assumptions as we did in our previous papers [29], [30], [31], [32], we interact with the environment and predict future user requests in the short term. Then, we adapt our caching decisions based on these predictions.

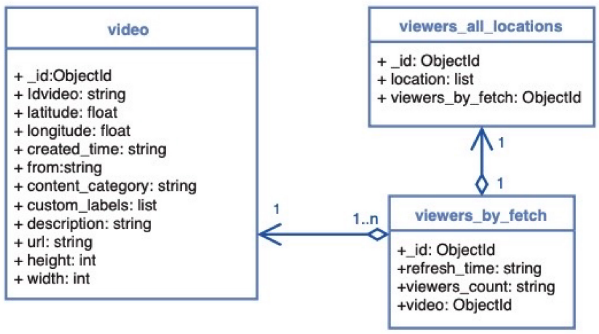


FIGURE 1. Structure of the Facebook dataset [33].

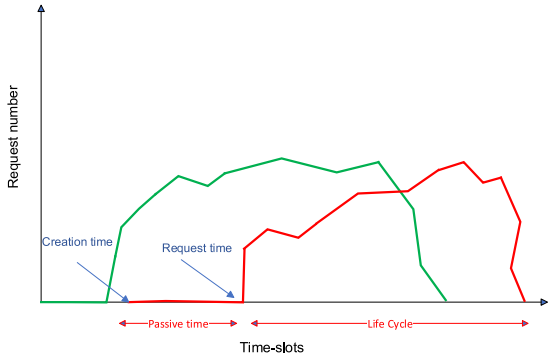


FIGURE 2. Request numbers for two videos vs time-slots.

III. DATASET USED AND PRE-PROCESSING

A. DATASET CHARACTERISTICS

To verify the caching techniques on a real dataset we require one that includes the exact time of user requests and the number of user requests for each content. In this paper, we use the dataset of the Facebook Live Dynamic platform [33]. The content here is videos posted by users. Facebook Live allows verified users to upload content. Specifically, the data spans from March 23, 2018, at 13:38:20, to June 22, 2018, at 15:23:59. After the removal of outliers and incomplete data, the final dataset comprised 644,938 distinct files.

The structure of the dataset is represented in Fig. 1. Each video class is defined by its dataset *"id"*, the identifier given by Facebook *"idvideo"*, the *"latitude"* and *"longitude"* of the broadcaster location, the broadcasting time called *"created time"*, and finally, the *"height"* and the *"width"* of the content, which serve to determine the quality of the video. It also has a *"refresh time"* which shows the request time and *"viewers count"* which represents the number of requests for each content.

To extract the information and build up the dataset, we start with the file *viewers-by-fetch*, which has the information on request times. We then search for the associated ID of the video in the *viewers-by-fetch* file and the *video* file to attach video features such as category and location of content creation. To determine the location of the users requesting the videos, we search for the ID from the *viewers-by-fetch* file and then in the *viewers-all-locations* file to obtain the location and number of requests for each video. This process allows us to refine the dataset by adding relevant features that improve the training process and the overall predictive performance. We cannot mathematically model the user requests of this dataset as the file popularities of the videos do not follow any known models. Files are created at a time, requested by users and then requests for that specific video die out. In other words, the requests in this case have life cycles and they are neither stationary nor piece-wise stationary. An illustration for the case of two videos can be found in Fig. 2. As can be seen, a video is created, after a period of time, which we call *passive time*, is requested by users and after passing its life cycle, no one requests that file again.

TABLE 1. Description of dataset features - original group.

Feature Name	Description
Video id	A unique name which shows the video id
Longitude	Longitude of the content creator
Latitude	Latitude of the content creator
Refresh time	The exact time of being requested
Creation time	The exact time of being created
Height	The height of the content
Width	The width of the content
Content category	A string specifying the content category
Request count	The number of requests of the content
Description	Description of the content

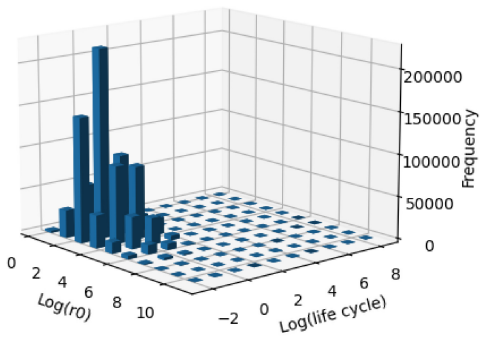


FIGURE 3. 3D histogram of log (r_0) and log(life cycle) in the dataset.

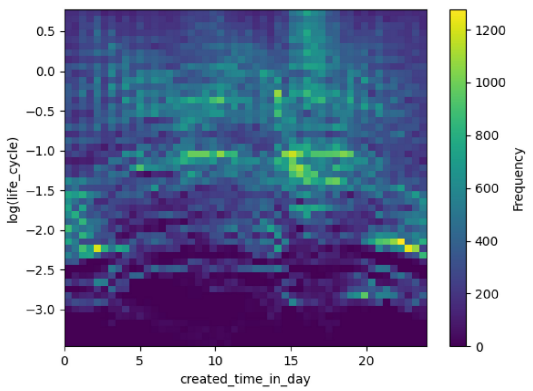
Table 1 summarizes the original features in the dataset and explains their characteristics. As can be seen, this dataset has the time of user requests ("Refresh time") and the number of requests ("Request count") for each specific content ("Video id"). The other features add extra information to assist us in making caching decisions based on predicted future requests.

B. DATA PRE-PROCESSING

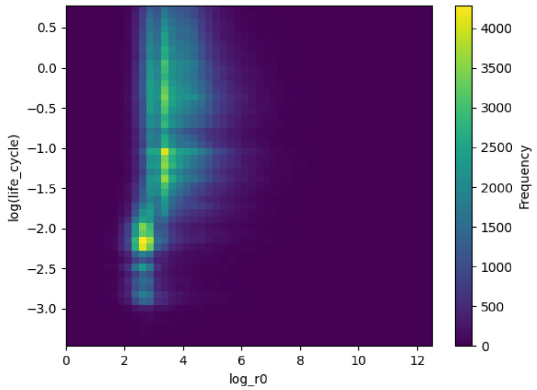
In this section, we explain the pre-processing of the dataset to prepare the data for further predictions. Importantly, we derive some new features to help in the learning process.

1) FEATURE EXTRACTION AND AUGMENTATION

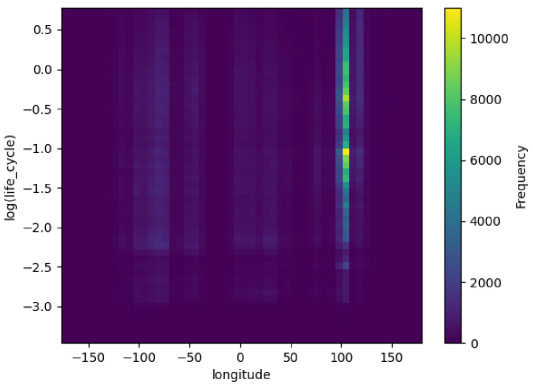
Table 2 summarizes the features derived to help in the learning process.



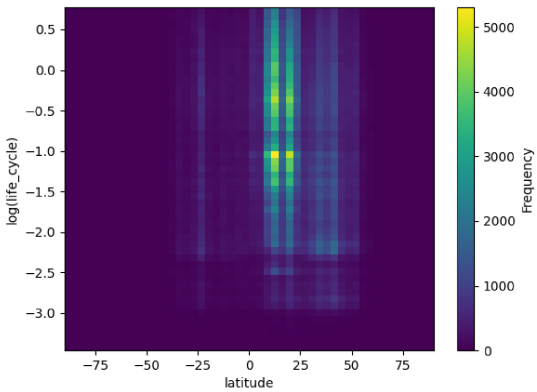
(a) Joint histogram of created time in a day and log(life cycle)



(b) Joint histogram of log(r0) and log(life cycle)



(c) Joint histogram of longitude and log(life cycle)



(d) Joint histogram of latitude and log(life cycle)

FIGURE 4. Various histograms representing different aspects of the data.

TABLE 2. Description of dataset features - created group.

Feature Name	Description
Content category number	Assigned number to the content category
Life cycle	The life cycle of the content
Passive time	The difference of creation time and refresh time in the scale of 5 minutes
Created time in a day	Relative time of creation on the scale of 5 minutes (0-288)
Requested time in a day	Relative time of request on the scale of 5 minutes (0-288)
First number (r_0)	The first number of request for the content

The introduction of new features serves to add meaningful information to the dataset and ensure the feasibility of predictions. For instance, "Life cycle", representing the total time slots during which a specific video has an active refresh time, provides valuable additional information. On the other hand, "Content category number" converts string categories into numerical values, making them suitable for prediction algorithms. Another feature, "First number (r_0)", represents the first number of requests for a video, as we believe the number of initial requests can indicate potential popularity.

Additionally, features such as the "Requested time in a day" and "Created time in a day" are defined with a nighttime reference, focusing on the time of day rather than the specific date of the requests. "Passive time" is another feature that shows the time difference between a file's creation and its first request, further enriching the dataset for more accurate predictions. All time scales are based on a 5-minute interval. This choice unifies the dataset's refresh time duration, which varies between 4 to 5 minutes and simplifies processing. The interval can be adjusted without loss of generality, though the duration impacts prediction performance; longer intervals may reduce sensitivity to new requests.

Having described the features collected, we analyze these features to illustrate how they capture information about the dataset. Fig. 3 shows the empirical frequency of different content, using a logarithmic scale, of r_0 and life cycle which approximates the joint probability distribution function of these features. Since these features do not have equal probabilities across all instances, we should be able to learn the features' contribution in predicting the user requests and life cycle. In other words, This non-uniformity is beneficial because it suggests that there are patterns or trends within the data. Fig. 4 represents the joint histograms of different pairs of features. The subfigures are for visualization purposes

to confirm that there is no visible correlation between features. As a consequence, we might not be able to extract information and make predictions with basic linear regression methods.

Another important pre-processing step is to ensure that all the features that participate in the learning process are numeric. This is easily accomplished for the time-based features which are converted to numeric values (as mentioned, with a scale of 5-minute blocks). Content categories are word descriptions of the files' content. Initially, we had 1174 different categories. To convert content categories to numerical vectors we use the Google 300 NLP Model (Google News Word2Vec), a pre-trained model developed by Google [34]. Each word in the Google News Word2Vec model is represented by a 300-dimensional vector. To avoid unnecessary complexity and large dimensions, we use principal component analysis (PCA) for dimensionality reduction [35]. Here, we use the first principal component, which represents the direction of maximum variance in the data as the numeric representation of content categories.[1] The reason we do not use simpler coding methods, such as one-hot encoding, is that one-hot encoding does not necessarily assign similar codes to similar categories. This can degrade the training process, as the category numbers are selected based solely on their order of appearance.

2) HANDLING MISSING DATA

In the available dataset, there are two types of missing data: missing a few time slots and missing some days. For the first category, to make use of all the available data, we assume that the content has been requested but the requests are not recorded. In these cases, we use linear interpolation to fill the missing time slots with neighboring numbers [36].

The other missing data belongs to the group in which we do not have any record of the content requests for days. In these cases, we do not know whether there have been no requests for the content or the information is censored. Censored data refers to information that is incomplete due to an observation period ending or data not being recorded, rather than the absence of the underlying event (e.g., content requests). We treat this portion of data as censored data, where the true duration of content requests is unknown; the content might have ceased being requested or continued beyond the last observation.

This censored data must be identified and treated appropriately to accurately predict survival probability, as it does not provide complete information on the content's life cycle. However, the data representing the period when requests were observed can still contribute valuable insights for training our pattern prediction section. In other words, each existing segment within the time series can be employed to predict the subsequent existing segment, ensuring that predictions are made only within the bounds of the available data. However, the truncated data lacks information about the life cycle, as the cut data does not include details on when the request sequence ends.

[1] Using only the first principal component was a choice; we do not believe adding more dimensions changes our contributions in any material way.

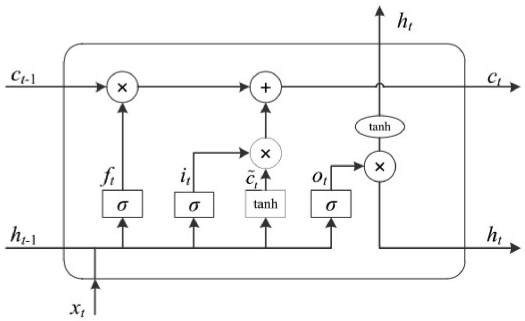


FIGURE 5. LSTM unit [18].

3) REMOVING OUTLIERS AND ERRORS

We identify and remove outliers from our dataset, ensuring the integrity and reliability of our analysis. First, we calculate the interquartile range (IQR) for each variable, which is the difference between the 75th and 25th percentiles [37]. We then define outliers as those data points that fell outside the bounds of 1.5 times the IQR below the 25th percentile and 1.5 times the IQR above the 75th percentile. By applying this widely accepted statistical technique [38], we effectively filter out extreme values that could bias our results.

In addition, the dataset has some data points that are not correct. For example, for out-of-range longitudes and latitudes, we cut them and push them back to the boundaries.

4) NORMALIZATION

Our feature space comprises different variables with different scales, from the number of requests to time slots and width. To unify all of them, we use a common range between zero and one. In this regard, for each feature, we scale the feature based on the maximum and minimum of that feature in the whole dataset to the range that we need, which is called Min-Max scaling technique [39].

IV. PROPOSED FILE POPULARITY FORECASTING

LSTM networks address the challenge of incorporating long-term dependencies in RNNs through a specialized design. This architecture introduces a memory cell, represented as c_t, which operates outside the traditional RNN data stream by adding an auxiliary state unit [40]. We briefly review LSTM networks to illustrate their use in our application.

As depicted in Fig. 5, LSTMs partition the hidden state of the RNN into two components: the memory cell c_t and the active memory h_t. The duty of the memory cell is to conserve the sequential input features, with its previous state modulated by a forget gate f. The active memory h_t acts as the output, with its proportion of the current memory c_t governed by the output gate o. The input gate i is responsible for managing how much of the present state information h_{t-1} and input x_t is stored in the memory cell. These

gates are not static; they adjust their responses based on a nonlinear activation after linearly combining the preceding state information h_{t-1} and the current input x_t. The LSTM framework is mathematically formulated as follows:

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right) \quad (1)$$
$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right) \quad (2)$$
$$\tilde{c}_t = \tanh\left(W_c \cdot \left[h_{t-1}, x_t\right] + b_c\right) \quad (3)$$
$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t \quad (4)$$
$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t\right] + b_o\right) \quad (5)$$
$$h_t = o_t \times \tanh(c_t) \quad (6)$$

where W_f, W_i, W_c, and W_o denote the weight matrices, and b_f, b_i, b_c, and b_o are the bias vectors. The new candidate state $\tilde{c}_t$ is formulated through a **tanh** layer processing x_t and h_{t-1}. The sigmoid function, $\sigma(\cdot)$, constraints outputs within the range of 0 to 1, thus facilitating gate operations that resemble binary decisions in controlling information flow.

An LSTM forecasting method consists of several LSTM units in which the weights are updated based on the training data and cost function. The network attempts to find the optimum weights for carrying the past, forgetting, making predictions and forwarding the results to the output. These weights can be controlled by the cost function, representing the ultimate goal for the updating process.

In our dataset, the input sequences consist of user requests over time. As users request different content for variable times, these input sequences vary in length, presenting a challenge for LSTM networks, which typically require uniform sequence lengths for processing. To address this variability, we use zero padding for shorter sequences to match the length of the longest sequence within a batch. While this approach facilitates the operational requirements of neural network libraries, it introduces the problem of skewed computations, as the padded values are not representative of any meaningful data and can adversely affect the network's learning process.

To mitigate this issue, we use masking. Masking is a process through which the LSTM network is instructed to selectively ignore the padded values during training and inference [41]. This is achieved by constructing a binary mask that parallels the input data tensor. Each element of this mask correlates with the corresponding element in the input tensor, with a binary value of 1 indicating a valid data point and 0 signifying a padded value.

When integrated into the LSTM architecture, this binary mask operates as a filter during the forward and backward propagation phases. The LSTM units utilize the mask to bypass operations on time steps where the mask value is 0, ensuring that the padded entries do not contribute to the loss function and have no influence on the weight update process during backpropagation. Furthermore, for sequence-wide computations such as averaging or summarizing, the mask ensures that only the actual sequence data is considered. With masking, the LSTM network ensures that it does not learn from padded values, which are not real data. It focuses solely on sequences with meaningful temporal dependencies and facilitates efficient batch processing, making the use of masking highly advantageous.

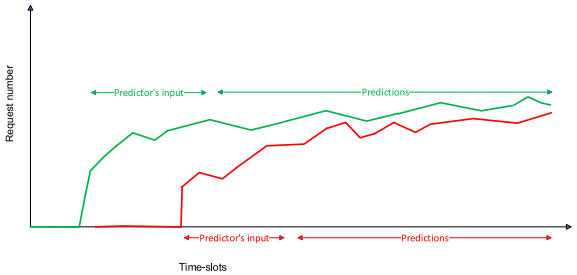


FIGURE 6. Two sample LSTM predictions of user requests.

In our dataset, as the content requests are time series, we use an LSTM network to detect patterns over time and do forecasting. We train an LSTM block such that we provide the network video features and some consecutive requests over time and predict the upcoming requests. It should be noted that here, we just predict the request number and not the life cycle. In other words, if we do not take into account that the requests might die after a while, the prediction is not accurate. Hence, this stage is the first stage which must be combined with the survival analysis information to make accurate decisions. Fig. 6 shows sample outputs of the LSTM predictor which follows the pattern *without accounting* for the life cycle.

As the inputs of our LSTM network, we use the following features from Table 1: "Longitude", "Latitude", "Height", "Width", "Request count", and from Table 2: "Content category number", "Passive time", "Created time in a day", "Requested time in a day", and "First number r_0". We skip some features from Table 1, such as "Video id", "Content category", and "Description", as they are not numeric. Instead of using "Refresh time" and "Creation time", we utilize the relative time-related variables introduced in Table 2. In addition, as we want to make predictions at any time even in the middle of an ongoing time series, we do not include life cycle information as a feature which has information of the future and makes the prediction non-causal. The input vector for our LSTM model includes the listed features and the past "Request Count" for each specific video up to the current time, resulting in a vector length of 27. This length allows us to include the maximum lengths of the time series after removing outliers. For videos with insufficient past data, we use the masking technique to zero-pad the vector elements, later removing the effect of zeros from the predictions.

Table 3 details our LSTM architecture. Since the network needs a fixed-size input, we consider the maximum length of the user requests (the one with the maximum life cycle) in addition to the other features as the input size. In the cases where the life cycle is lower, we use the masking technique with zero padding as described. A batch size of 32 means that we update the LSTM weights after every 32 samples

TABLE 3. LSTM network architecture.

No	Characteristics	Description
1	Sequence Input	With 27 dimensions (maximum of the time series length + other features)
2	LSTM	50 units
3	Loss function	The ranking difference percentage in real and predicted data
4	Batch size	32
5	Epochs	100
6	Training data	80%

which are trained in parallel. Then as these samples are picked randomly from the data, we repeat the training 100 times.

An important factor that is different in our modeling is that we define a custom loss function which calculates the rank difference. Since our application is in caching with a limited cache capacity, we are interested in the most popular files. In this regard, the numerical value of the requests is less important than the file's rank in a ranked list of popularity; we, therefore, want to highlight the rank in our loss function. Our loss function accounts for the number of files that are mislocated in the predictions, i.e., it counts the files that receive an incorrect rank after the prediction.

If we define $\hat{\boldsymbol{y}}_t = [\hat{y}_{t,1}, \ldots, \hat{y}_{t,N_t}], t \in \{1, \ldots, T\}$ as the predicted output of our LSTM network and $\boldsymbol{y}_t = [y_{t,1}, \ldots, y_{t,N_t}], t \in \{1, \ldots, T\}$ as the vector of actual output values and N_t as the total content number at time t, we define our loss function as[2]

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{\sum_{t=1}^{T} \sum_{i=1}^{N_t} |g(\hat{y}_{t,i}, \hat{\boldsymbol{y}}_t) - g(y_{t,i}, \boldsymbol{y}_t)|}{\sum_{t=1}^{T} N_t}, \tag{7}$$

where, $g(a, \boldsymbol{a})$ is a function that calculates the rank of the input element a in the vector $\boldsymbol{a}$ and $\boldsymbol{\theta}$ is the learning parameter vector. Fig. 7 represents the flowchart of our request evolution predictor with all the details of LSTM network for $T = 3$ (predicting 3 upcoming time slots). The algorithm also adds the survival probability predictions and transfers all the information to the reward calculator (which we will explain in the following sections). The input to the LSTM network shows all the pre-processing stages, the hidden layer consists of several LSTM units optimizing the output with an Adam optimizer and our custom loss function.

V. PROPOSED SURVIVAL TIME ANALYSIS

In this section, we use survival analysis to predict the life cycle of content requests. Survival analysis is about predicting the time duration until an event occurs, and in our case, the event is the death of specific content. The hazard function is defined as the instantaneous rate at which events occur, at a specific time, given that the event has not occurred before that time. This means it measures the likelihood of the event happening at a particular moment, assuming it has not yet happened [42]. Considering T_E as the random time-to-event variable and $\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,p}]^T$ as specific content i with p features, the hazard function at time t given $\boldsymbol{x}_i$, is expressed as:

$$h(t|\boldsymbol{x}_i) = \lim_{\Delta t \to 0} \frac{P(t \le T_E < t + \Delta t | T_E \ge t)}{\Delta t}. \tag{8}$$

Here, time-to-event is defined as the time remaining until there is no request for the content or in other words, the content is dead. Eq. (8) measures the probability of the event happening at the next Δt time, knowing that it has not occurred until now.

The survival probability given content $\boldsymbol{x}_i$, $P_s^{t,i}(t|\boldsymbol{x}_i)$, is the probability that the time of an event, T_E, is later than time t, conditional on x_i. The survival probability can be formulated as [43]:

$$P_s^{t,i}(t|\boldsymbol{x}_i) = \exp\left(-\int_0^t h(u|\boldsymbol{x}_i)\, du\right), \tag{9}$$

where $h(u|\boldsymbol{x}_i)$ is the hazard at time u given covariates $\boldsymbol{x}_i$. In the CPH approach, the hazard function at time t is modeled as [44]:

$$h(t|\boldsymbol{x}_i) = h_0(t) \exp\left(\boldsymbol{\beta}^T \boldsymbol{x}_i\right), \tag{10}$$

where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_p]^T$ is a vector of regression parameters, and $h_0(t)$ is the baseline hazard function, assumed to be not a function of $\boldsymbol{x}_i$ making the model semi-parametric [45].

The regression parameters in the CPH can be estimated by minimizing the negative log partial likelihood cost function [46]:

$$l(\boldsymbol{\beta}) = -\sum_{i=1}^{n} E_i \left(\boldsymbol{\beta}^T \boldsymbol{x}_i - \log \sum_{j \in R(t_i)} \exp\left(\boldsymbol{\beta}^T \boldsymbol{x}_j\right)\right), \tag{2}$$

where n is the number of files, t_i is the censored or observed survival time for content i, and E_i is an indicator for whether the survival time is censored (no Event) ($E_i = 0$) or observed ($E_i = 1$). We note that, if a part of the data is missing and there is no request for the content which was requested before the missing portion, it is not evident when exactly the content has died, i.e., is censored. $R(t_i)$ is the risk set at time t_i, which is the set of all the content which are still requested at time t_i.

[2]It should be noted that as the LSTM optimizer uses gradient descent to adjust the weights, we need to make this loss function differentiable. In this regard, with the aid of a Softmax function, we introduce a *Soft* ranking function to implement our LSTM network.

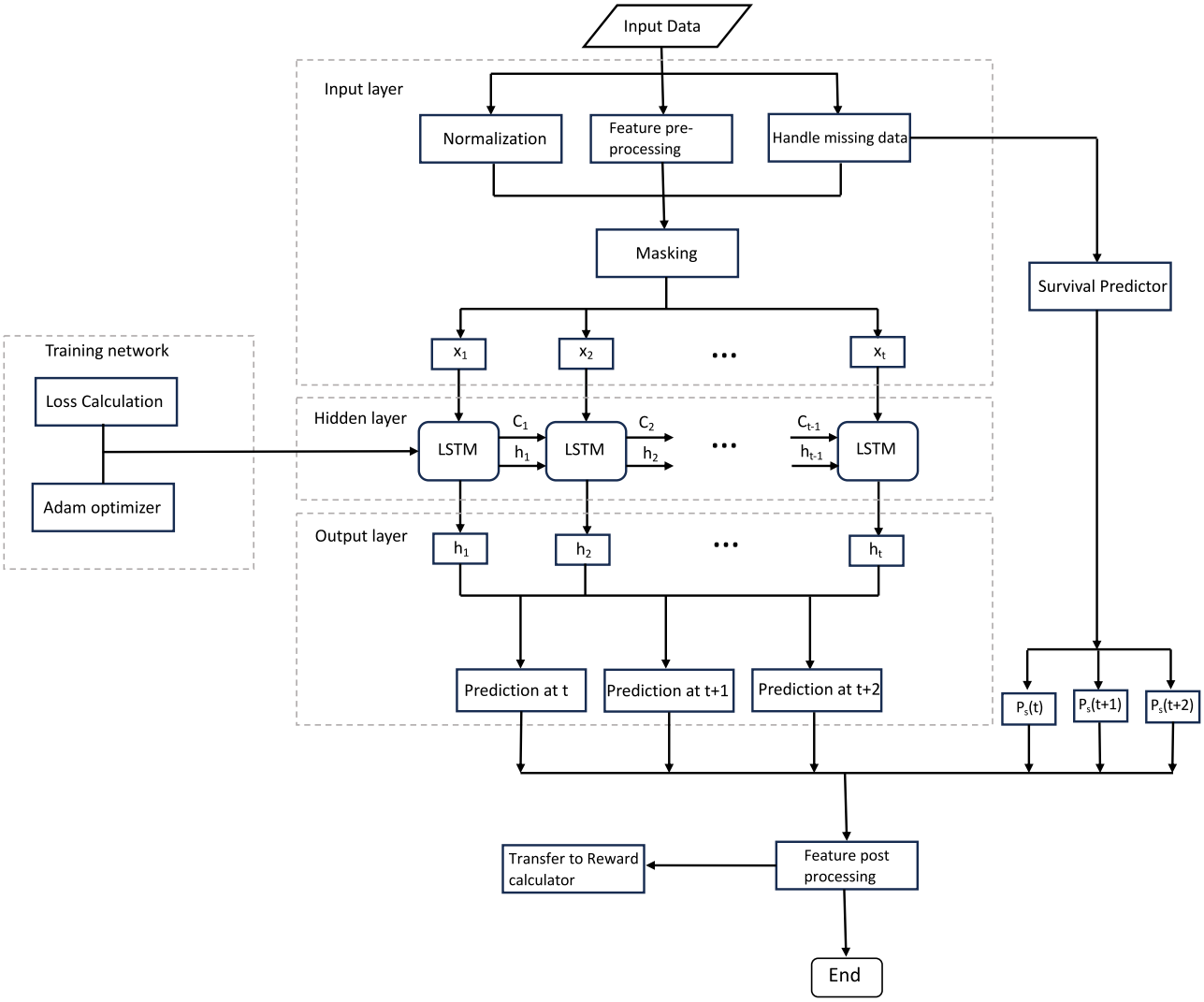


FIGURE 7. Flowchart of the proposed algorithm.

To compare the performances in survival probability prediction, the concordance index (CI) is used as an evaluation metric [47]. The CI is defined as follows:

$$\text{CI} = \frac{\#\text{correctly ordered pairs}}{\#\text{all possible ranking pairs}}. \tag{11}$$

This equation is the fraction of all pairs of subjects whose survival times are correctly ordered among all possible ranking pairs [47]. The value of 0 is with the worst performance, 0.5 is a random guess and 1 is the perfect condition.

A. OUR PROPOSED E-CPH MODEL

In this section, we explain our proposed extensions to the CPH approach, referred to as our E-CPH model. To improve the CPH performance, as it relies on the regression between the content features, we extend the feature dimensions to capture more information. We note that here instead of mainly relying on the previous time slots' content requests to predict future requests (which was the LSTM-based approach), we rely on the content's characteristics to be able to predict its survival rate.

In the E-CPH framework, instead of coding content categories as scalar numbers, we map them to a higher dimensional space to capture as much information as possible. As we mentioned, to convert content categories to informative vectors we use the Google 300 NLP Model. Then, with PCA we lower the dimension, but unlike in the LSTM case, a single dimension, we keep more components as long as they can be informative.

Fig. 8 illustrates our approach to preparing data as input to our E-CPH model. In our approach, we employ PCA to reduce the dimensionality of our dataset while retaining significant information. Specifically, we select the number of principal components such that the cumulative variance they explain reaches 90%. In practical terms, this leads to a reduction from 300 features to 50 principal components in our dataset.

TABLE 4. Content characteristics at time 2018-05-18 13:28 - 13:33.

Content name	Age at the time	Passive time	Survival time	Event
”5afe9511fce45733149362fb”	24	1	0	1
”5afe959efce4573314936473”	23	1	0	1
”5afe98d7fce4573314936d37”	21	6	4	1
”5afe9a46fce4573314937139”	19	1	6	1
”5afe9dddfce4573314937a6a”	16	1	3	1
”5afe9df3fce4573314937b38”	16	2	0	1
”5afea480fce4573314938c96”	2	0	14	1
”5afeb035fce457331493ad19”	1	3	2	1
”5afeaeb9fce457331493a92a”	2	1	1	1
”5afeaeb8fce457331493a91e”	2	9	2	1
...	...	...	...	...

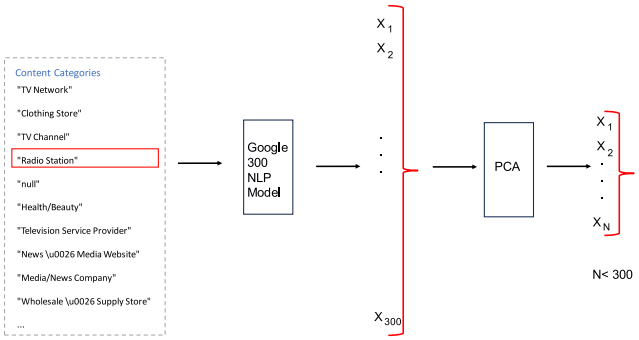


FIGURE 8. Data pre-processing for E-CPH.

Table 4 presents characteristics of the input data for our E-CPH model at a given time slot, including content requested by users, the content's age, its life cycle, and whether the data point is censored (Event = 0). Note that the features used for life cycle prediction are the same ones utilized as inputs for the LSTM network and those presented in Table 4.

It is important to note that, for the sake of clarity, this table does not display the 50-element vector resulting from PCA or other specific features of each content item.

VI. POMDP-BASED CACHING

Having described the data preprocessing steps, the use of LSTMs to predict requests and the E-CPH approach to estimate content lifetimes, we are ready to present our caching strategy that exploits these developments.

We consider content requests arriving through discrete time blocks of 5 minutes with a caching decision cycle of every T blocks. The environment, which is the user requests, constitutes a finite-state finite-horizon Markov process. The state set S is the set of cache states and user requests and A and Ω are the action and observation spaces, respectively. Considering the current and past features as observations, the content requests in the future as our network states can be variable.

In other words, the current observations lead to a *belief* of alive or dead content. In fact, in a POMDP, the agent does not have direct access to the true state of the environment due to its partial observability. Instead, the agent maintains a belief $b(s)$, which is a probability distribution over the set of possible states S. The belief $b(s)$ represents the agent's degree of confidence that the environment is in state s, given the history of actions and observations. b_0 is an initial PMF over states such that $b_0(s)$ is equal to the probability that the initial state of the system is s, which indicates the initial belief. $R(\cdot,\cdot)$ is a bounded reward function, such that $R(s,a)$ is the real-valued reward obtained executing action a in state s. Hence, we can define our finite-horizon POMDP as a tuple (S, A, Ω, R, b_0) [48].

We formulate our caching problem with a POMDP, in which the hidden state is the state of the content. At each time slot, we cannot be sure if the file remains alive for the upcoming time slots or if it dies. In this regard, the survival probabilities as the outputs of our E-CPH are used as the inputs to our POMDP. POMDPs usually learn the belief states within the learning process and update the popularities. However, in our case, as the network is not stationary, we use our previously proposed methods to extract the life cycle information as belief. In other words, we help the POMDP problem with extra belief information to learn a non-stationary environment.

In E-CPH, the output is the vector of survival probabilities for each content for the upcoming T time slots. We also have the prediction vector from LSTM for the next T time slots, each the predicted content popularity at time t is $\hat{\boldsymbol{r}}_t = [\hat{r}_{t,1}, \ldots, \hat{r}_{t,N_t}]$. We aim to make caching decisions for every T time slots. In other words, we update our cache content every T time slots. Our objective is to maximize cache hit, which is the number of requested content at each time that can directly be served by the cache memory.

If we define the number of cache hits as $H(s_{t,i}, a_{t,i})$ for the content state $s_{t,i} \in \{\text{alive, dead}\}$ and action $a_{t,i} \in \{0, 1\}$ indicating whether we store the file (1) or not (0), $H(s_{t,i} =$

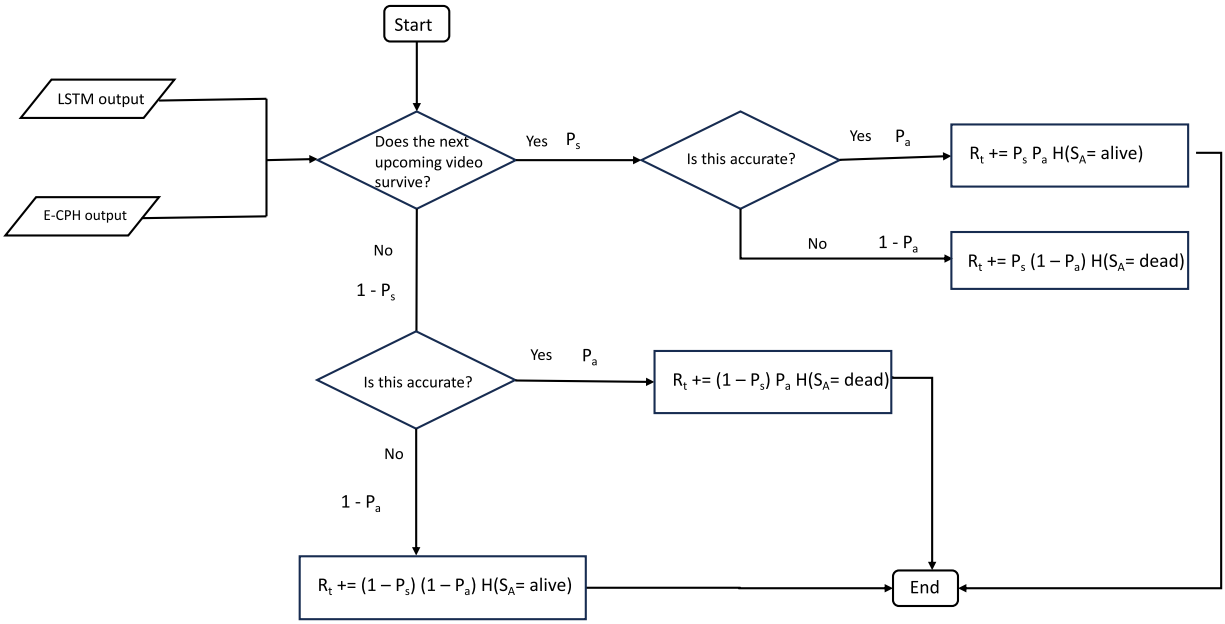


FIGURE 9. Flowchart of the proposed POMDP-based reward calculator for content *A*.

alive, $a_{t,i} = 1$) can be written as

$$H\big(s_{t,i} = \text{alive}, a_{t,i} = 1\big) = \max_{\{\delta_{t,j}\}} \left\{ \sum_{j=1, j\neq i}^{N_t} \hat{r}_{t,j}\delta_{t,j} + \hat{r}_{t,i} \right\}$$
$$\text{s.t.} \quad \delta_{t,j} \in \{0, 1\} \ \forall j \in \{1, \ldots, N_t\},$$
$$\sum_{j=1}^{N_t} \delta_{t,j} = C_c - 1, \tag{12}$$

where C_c is cache memory capacity and $\delta_{t,j}$ is an indicator for each content and takes an integer value showing the existence of content j in the cache at time t. If content i is alive and we take the action of storing it, it adds $\hat{r}_{t,i}$ to the total cache hits. For the remaining vacant spots in the cache memory, we store the $(C_c - 1)$ top candidates.

Similarly, $H(s_{t,i} = \text{dead}, a_{t,i} = 1)$ can be defined as

$$H\big(s_{t,i} = \text{dead}, a_{t,i} = 1\big) = \max_{\{\delta_{t,j}\}} \left\{ \sum_{j=1, j\neq i}^{N_t} \hat{r}_{t,j}\delta_{t,j} \right\}$$
$$\text{s.t.} \quad \delta_{t,j} \in \{0, 1\} \ \forall j \in \{1, \ldots, N_t\},$$
$$\sum_{j=1}^{N_t} \delta_{t,j} = C_c - 1. \tag{13}$$

As can be seen, when the state of the file in the next time slot is dead, even if the action was storing the content, it does not have any contributions to the total number of cache hits. This is because we do not wish to cache content we believe to be dead.

For every content i, the reward $R_{t_r,i}(b_{t,i}, a_{t,i} = 1)$ at a reference time t_r for the upcoming T time slots is defined as

$$R_{t_r,i}\big(b_{t,i}, a_{t,i} = 1\big) = \sum_{t=t_r}^{t_r+T-1} \sum_{s_i \in \{\text{alive, dead}\}} b(s_i, t) H\big(s_{t,i}, a_{t,i} = 1\big). \tag{14}$$

For each content i, assuming $P_s^{t,i}$ as the survival probability of time t and P_a as the corresponding accuracy for calculating the survival probability, $b(s_i, t)$ can be calculated as

$$b(s_i, t) = \begin{cases} P_s^{t,i} P_a + \big(1 - P_s^{t,i}\big)(1 - P_a) \text{ if } s_i = \text{alive}, \\ P_s^{t,i}(1 - P_a) + \big(1 - P_s^{t,i}\big) P_a \text{ if } s_i = \text{dead}. \end{cases} \tag{15}$$

Fig. 9 at, represents the reward calculation for each time slot after receiving the predicted values of the LSTM network and survival probabilities of E-CPH for a sample file, here called content "A". As can be seen, the first step is checking whether the file will survive the next time slot, which is True with probability P_s and False with probability $1 - P_s$. Then, for each case, we check whether the calculated probability is accurate or not. Finally, the total reward for the next time slot t is the summation of all the rewards.

Algorithm 1 shows the steps to calculate the action $\boldsymbol{a}^*_{t_r} = [a^*_{1,t_r}, \ldots, a^*_{N_{t_r},t_r}]$ which is the caching decisions for the N_{t_r} files for time slot t_r for the upcoming T time slots. According to the algorithm, we start from a reference time t_r, knowing the cache capacity C_c and the output of the LSTM network, $\{\hat{\boldsymbol{r}}_{t_1}, \hat{\boldsymbol{r}}_{t_2}, \hat{\boldsymbol{r}}_{t_3}, \ldots, \}$, and the E-CPH survival probabilities, $P_s^{t,i}, P_a$ for $t \in \{t_1, t_2, \ldots, \}$. For each T time slots, we calculate the reward of caching each of the $N_t, t \in \{t_r, \ldots, t_r + T - 1\}$ files. Then, we decide to cache C_c number of files that have the highest reward values.

Algorithm 1: POMDP-Based Cache Decision Making

Input: $\{\hat{\boldsymbol{r}}_{t_1}, \hat{\boldsymbol{r}}_{t_2}, \ldots\}$, $P_s^{t,i}$, P_a for $t \in \{t_1, t_2, \ldots\}$, C_c, T
Output: $\boldsymbol{a}_{t_r}^*$
for $t_r \leftarrow t_1$ **to** ... *for every T time slots* **do**
  $R_{t_r} \leftarrow 0_{N_{t_r}}$
  for $i \leftarrow 1$ **to** N_{t_r} **do**
    $R_{t_r}[i] = R_{t_r,i}(b_{t_r,i}, a_{t_r,i} = 1)$
  $\boldsymbol{a}_{t_r}^* = \arg\max\{\boldsymbol{R}_{t_r}^T \boldsymbol{a}_{t_r}\}$
  s.t. $a_{i,t_r} \in \{0, 1\}$ for $i \in \{1, \ldots, N_{t_r}\}$
    $\sum_{i=1}^{N_{t_r}} a_{i,t_r} = C_c$

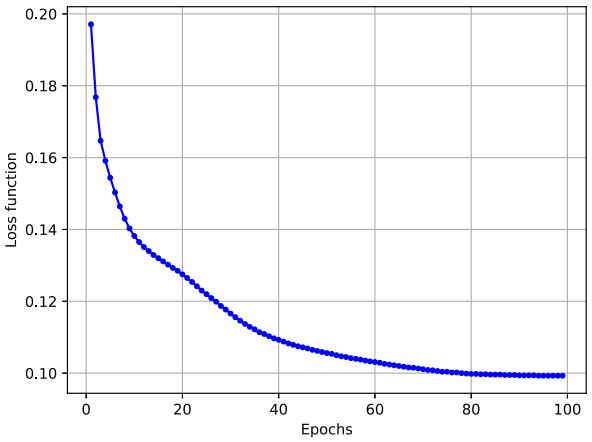


FIGURE 10. Loss function reduction in our LSTM training versus different epochs.

VII. SIMULATION RESULTS

In this section, we present the numerical results of our proposed predictive caching algorithm.

Fig. 10 shows the loss function reduction in our LSTM network with the characteristics explained in Table 3. The network predicts the popularity for three upcoming time slots ($T = 3$), a choice made to balance prediction accuracy and caching complexity. In general, the prediction can be extended to more time slots. As can be seen, the loss function decreases with an increasing number of epochs, which shows the effectiveness of training. Our loss function is the average file rank difference. After 100 epochs, we achieve an average rank difference of 0.1. Assuming 500 content requests at a time, the loss function implies that approximately 50 files are given incorrect rankings. We note that this can be either one dislocated video for a rank difference of 50 or several videos which are dislocated for more than one level for a maximum of 50 videos dislocated each for just one level.

Fig. 11 presents the results of training the LSTM network with the available dataset. The figure shows the sorted number of requests in one time slot of 5 minutes and compares it with the predicted values. The x-axis shows different content and as can be seen, the distribution is similar to a Zipf distribution which is one of the most common distributions for user requests. In addition, the predicted curve shows that the algorithm is trying to keep the ranking

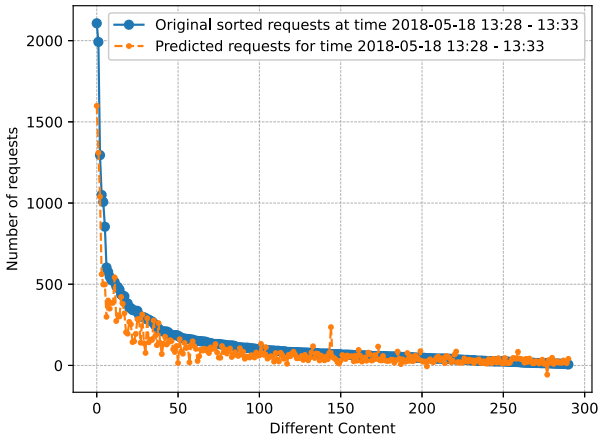


FIGURE 11. Number of requests for different videos in one time slot for the real and predicted requests.

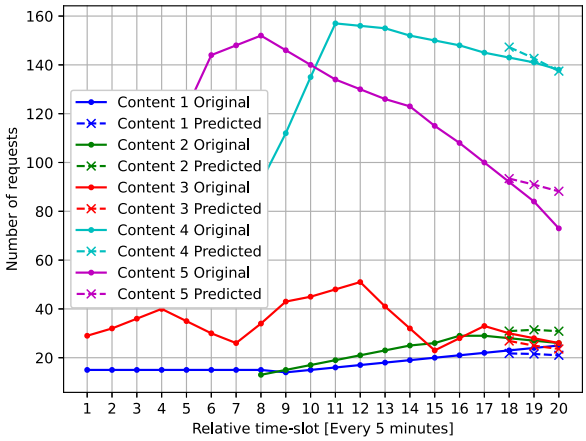


FIGURE 12. Samples of content request predictions for three upcoming time slots.

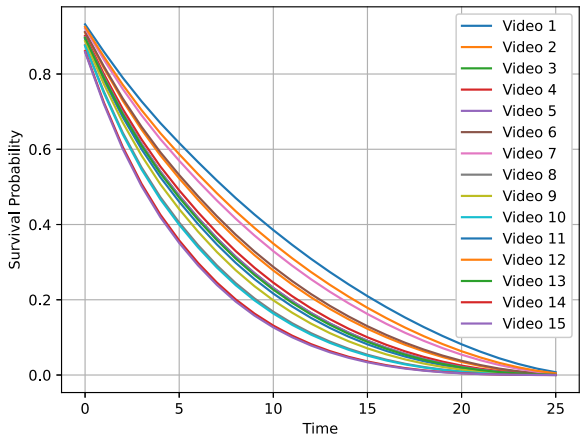


FIGURE 13. Survival probabilities of different videos at one time-slot over time.

orders with the original values as much as possible. However, some jumps and drops that suggest order dislocation are visible.

Fig. 12 represents 5 samples of content prediction as the output of our LSTM network. To avoid unnecessary information on the x-axis, we labeled the time slots with relative numbers, each showing periods of 5 minutes. In this

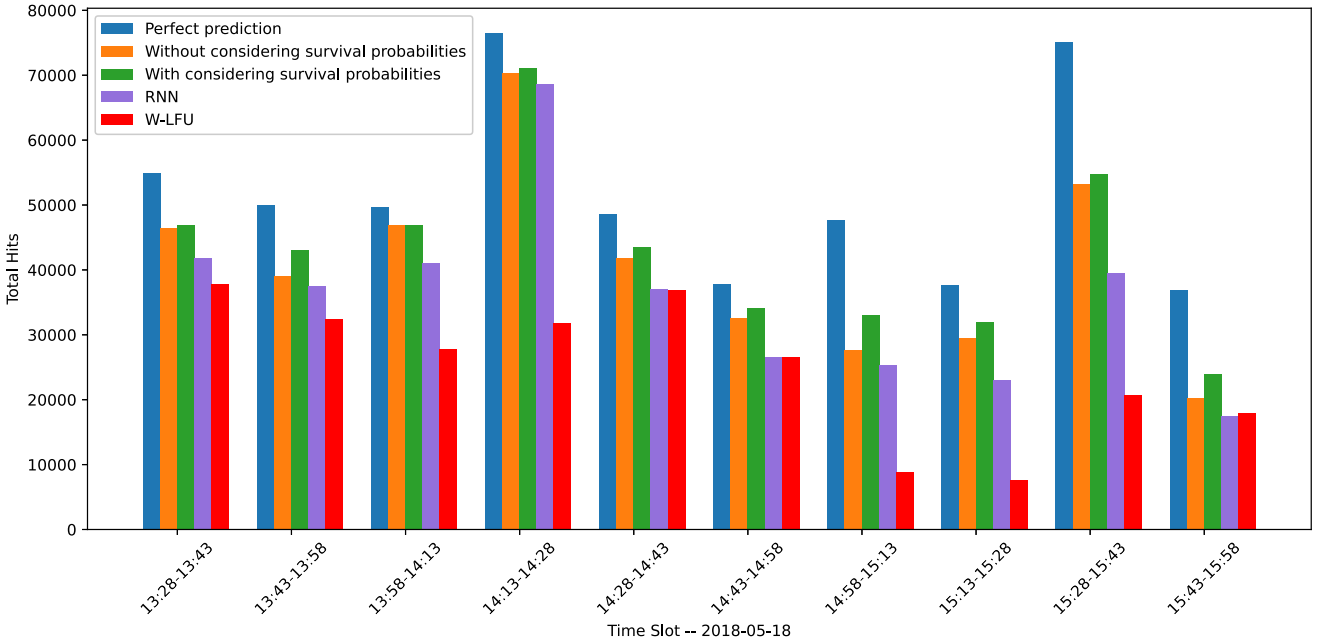


FIGURE 14. Cache hit versus time for different caching methods.

figure, we made a prediction at time slot 17 for the upcoming slots of 18, 19 and 20. As can be seen, some content has a lower duration which shows different times of requests. The dashed curves represent the predicted values of each content, plotted in the same color as the original ones. As can be seen, in a large part, the predicted values follow the user request patterns, specifically the order of the files.

Fig. 13 shows the probability of survival for 15 randomly selected videos in our test dataset. As can be seen, different videos have different survival probabilities. Besides, the survival probability decreases over time which shows the limited life cycle of the videos. At each time slot, we calculate the survival probabilities of each video for the upcoming T time slots, $T = 3$ in our case, and feed it as the input of our POMDP caching optimizer. We note that the CI factor, defined in (11), with the CPH survival model is 0.54, but improves to 0.63 upon using our proposed E-CPH approach.

Fig. 14 compares the total number of cache hits in our proposed model and perfect prediction (which is an upper bound for the network cache hit) and the traditional caching strategy, window-based least frequency used (W-LFU), with $W = 10$ time slots and $C_c = 50$. It also compares the performance with an RNN-based predictive caching model that utilizes 50 units, three output layers, and the Adam optimizer as the LSTM network. In a perfect prediction, the network is non-causal and hence it has all the information about the future, even the request numbers for new content. Each time slot is 5 minutes and hence the caching decisions are made for every three time slots, i.e., 15 minutes. We note that since the cache capacity is limited, even perfect prediction does not result in a 100% cache hit rate.

It should be noted that the accuracy of survival probabilities is calculated based on averaging over all the test datasets. In this regard, if we predict the file is alive (its survival probability is more than 50%) but the file is dead, we consider it an error and then sum up all the errors and divide it by the total number of predictions. For our dataset and E-CPH the probability of accuracy is $P_a = 78\%$. As can be seen, both our proposed methods with and without considering survival probabilities, have a higher number of hits in comparison with W-LFU. In most cases, the RNN predictor demonstrates a higher hit rate compared to the W-LFU caching method, which lacks artificial intelligence capabilities. However, the RNN predictor still underperforms relative to our proposed methods. In the final time interval, the W-LFU method performs comparably to the RNN predictor and even performs slightly better in that specific time. This is likely due to a potential correlation with previous time slots in that interval, which can surpass the predictive power of artificial intelligence, especially considering the inherent randomness in AI algorithms.

When we introduce the survival probabilities, in almost all the cases, we experience hit improvement. The reason for this improvement is that when we have content survival information, the cache optimization strategy will save the available spots in the cache memory for the files that will survive with a higher probability. In this regard, the cache memory is less occupied with content which is no longer requested. As can be seen, in the period 13:58 to 14:13, survival probability does not improve the cache hit. It can be explained due to the accuracy error which is 22% and also the existence of censored data.

VIII. CONCLUSION

In this paper, we investigated the application of caching techniques through practical experiments with a real-world dataset. Working with the dataset required customized preprocessing steps. This dataset indicates that user requests change over time and have a lifecycle, suggesting that some content may no longer be requested. We introduced a POMDP-based approach focused on improving cache hits by predicting the users' request patterns. To do so, we designed an LSTM network to forecast user requests over time by treating past request numbers as time series data. Furthermore, recognizing the finite lifespan of the content, we introduced an E-CPH model for survival analysis to estimate how long the content remains active. We then combined the predictions from the LSTM and the survival probabilities as inputs of our POMDP-based caching strategy, enhancing the cache hit rate in the network. Our findings compellingly showcase the effectiveness of our predictive caching strategy in dynamically adjusting to evolving user popularity trends and enhancing cache hit rates. For future work, we plan to consider additional performance metrics such as latency, throughput, and system resource utilization to further evaluate the effectiveness of our prediction methods in making caching decisions and improving these metrics.

REFERENCES

[1] H. Yeganeh and J. Rostampoor, "Downlink user association and uplink scheduling for energy harvesting users in software-defined mobile networks," *Phys. Commun.*, vol. 28, pp. 11–18, Jun. 2018.

[2] I. Tomkos, D. Klonidis, E. Pikasis, and S. Theodoridis, "Toward the 6G network era: Opportunities and challenges," *IT Prof.*, vol. 22, no. 1, pp. 34–38, Jan./Feb. 2020.

[3] J. R. Bhat and S. A. Alqahtani, "6G ecosystem: Current status and future perspective," *IEEE Access*, vol. 9, pp. 43134–43167, 2021.

[4] N. Nomikos, S. Zoupanos, T. Charalambous, and I. Krikidis, "A survey on reinforcement learning-aided caching in heterogeneous mobile edge networks," *IEEE Access*, vol. 10, pp. 4380–4413, 2022.

[5] H. S. Goian, O. Y. Al-Jarrah, S. Muhaidat, Y. Al-Hammadi, P. Yoo, and M. Dianati, "Popularity-based video caching techniques for cache-enabled networks: A survey," *IEEE Access*, vol. 7, pp. 27699–27719, 2019.

[6] W. Hoiles, A. Aprem, and V. Krishnamurthy, "Engagement and popularity dynamics of YouTube videos and sensitivity to Meta-data," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 7, pp. 1426–1437, Jul. 2017.

[7] B. Shulman, A. Sharma, and D. Cosley, "Predictability of popularity: Gaps between prediction and understanding," in *Proc. Int. Conf. Web Soc. Media (ICWSM)*, 2016, pp. 348–357.

[8] G. Szabo and B. A. Huberman, "Predicting the popularity of online content," *Commun. ACM*, vol. 53, no. 8, pp. 80–88, 2010.

[9] H. Huang, Z. Wang, H. Zhang, X. Wang, C. Zhang, and W. Wang, "One for all: Unified workload prediction for dynamic multi-tenant edge cloud platforms," in *Proc. 29th ACM SIGKDD Conf. Knowl. Discov. Data Min. (KDD)*, 2023, pp. 788–797. [Online]. Available: https://doi.org/10.1145/3580305.3599453

[10] S. Tuli, G. Casale, and N. R. Jennings, "CILP: Co-simulation-based imitation learner for dynamic resource provisioning in cloud computing environments," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 4, pp. 4448–4460, Dec. 2023.

[11] A. Ioannou and S. Weber, "Exploring content popularity in information-centric networks," *China Commun.*, vol. 12, no. 7, pp. 13–22, Jul. 2015.

[12] Y. Zhang, X. Tan, and W. Li, "PPC: Popularity prediction caching in ICN," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 5–8, Jan. 2018.

[13] H. Nakayama, S. Ata, and I. Oka, "Caching algorithm for content-oriented networks using prediction of popularity of contents," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, 2015, pp. 1171–1176.

[14] N. Ben Hassine, D. Marinca, P. Minet, and D. Barth, "Popularity prediction in content delivery networks," in *Proc. IEEE 26th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, 2015, pp. 2083–2088.

[15] E. B. Abdelkrim, M. A. Salahuddin, H. Elbiaze, and R. Glitho, "A hybrid regression model for video popularity-based cache replacement in content delivery networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2016, pp. 1–7.

[16] M. Chen, W. Saad, C. Yin, and M. Debbah, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Trans. Wireless Commun.*, vol. 16, no. 6, pp. 3520–3535, Jun. 2017.

[17] A. O. Nwana, S. Avestimehr, and T. Chen, "A latent social approach to YouTube popularity prediction," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, 2013, pp. 3138–3144.

[18] Y. Yu, J. Cao, and J. Zhu, "An LSTM short-term solar irradiance forecasting under complicated weather conditions," *IEEE Access*, vol. 7, pp. 145651–145666, 2019.

[19] C. Zhang et al., "Toward edge-assisted video content intelligent caching with long short-term memory learning," *IEEE Access*, vol. 7, pp. 152832–152846, 2019.

[20] T.-V. Nguyen et al., "User-preference-based proactive caching in edge networks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2021, pp. 755–757.

[21] H. Mou, Y. Liu, and L. Wang, "LSTM for mobility based content popularity prediction in wireless caching networks," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[22] D. Li, H. Zhang, D. Yuan, and M. Zhang, "Learning-based hierarchical edge caching for cloud-aided heterogeneous networks," *IEEE Trans. Wireless Commun.*, vol. 22, no. 3, pp. 1648–1663, Mar. 2023.

[23] D. Li, H. Zhang, H. Ding, T. Li, D. Liang, and D. Yuan, "User-preference-learning-based proactive edge caching for D2D-assisted wireless networks," *IEEE Internet Things J.*, vol. 10, no. 13, pp. 11922–11937, Jul. 2023.

[24] M. Y.-K. Chua, F. R. Yu, and S. Bu, "Dynamic operations of cloud radio access networks (C-RAN) for mobile cloud computing systems," *IEEE Trans. Veh. Technol.*, vol. 65, no. 3, pp. 1536–1548, Mar. 2016.

[25] H. Kim, J. Park, M. Bennis, S.-L. Kim, and M. Debbah, "Mean-field game theoretic edge caching in ultra-dense networks," 2021, *arXiv:1801.07367*.

[26] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Trans. Mobile Comput.*, vol. 15, no. 8, pp. 1863–1876, Aug. 2016.

[27] P. Cong, K. Qi, and C. Yang, "Impact of prediction uncertainty of popularity distribution on proactive caching," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, 2019, pp. 747–752.

[28] S.-E. Elayoubi and J. Roberts, "Performance and cost effectiveness of caching in mobile access networks," in *Proc. 2nd Int. Conf. Inf.-Centric Netw. (ICN)*, 2015, pp. 79–88.

[29] J. Rostampoor and R. Adve, "Dynamic caching in a hybrid Millimeter-wave/microwave C-RAN," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2022, pp. 1–6.

[30] J. Rostampoor and R. S. Adve, "Optimizing caching in a C-RAN with a hybrid millimeter-wave/microwave fronthaul link via dynamic programming," *IEEE Trans. Commun.*, vol. 71, no. 2, pp. 923–934, Feb. 2023.

[31] J. Rostampoor, R. S. Adve, A. Afana, and Y. A. E. Ahmed, "CPRL: Change point detection and reinforcement learning to optimize cache placement strategies," *IEEE Trans. Commun.*, vol. 72, no. 4, pp. 2339–2353, Apr. 2024.

[32] J. Rostampoor, R. Adve, A. Afana, and Y. Ahmed, "Learning when and how to forget in variable window LFU caching," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, 2023, pp. 781–786.

[33] E. Baccour, A. Erbad, K. Bilal, A. Mohamed, M. Guizani, and M. Hamdi, "FacebookVideoLive18: A live video streaming dataset for streams metadata and online viewers locations," in *Proc. IEEE Int. Conf. Informat., IoT, Enabling Technol. (ICIoT)*, 2020, pp. 476–483.

[34] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
[35] S. Sehgal, H. Singh, M. Agarwal, V. Bhasker, and Shantanu, "Data analysis using principal component analysis," in *Proc. Int. Conf. Med. Imaging, m-Health Emerg. Commun. Syst. (MedCom)*, 2014, pp. 45–48.
[36] T. Blu, P. Thevenaz, and M. Unser, "Linear interpolation revitalized," *IEEE Trans. Image Process.*, vol. 13, pp. 710–719, 2004.
[37] K. Benkert, E. Gabriel, and M. M. Resch, "Outlier detection in performance data of parallel applications," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2008, pp. 1–8.
[38] J. Kim, "Using median as a threshold in determining anomaly in back-end authentication," in *Proc. 3rd Int. Conf. Comput. Intell. Appl. (ICCIA)*, 2018, pp. 249–254.
[39] H. W. Herwanto, A. N. Handayani, A. P. Wibawa, K. L. Chandrika, and K. Arai, "Comparison of min-max, Z-score and decimal scaling Normalization for zoning feature extraction on javanese character recognition," in *Proc. 7th Int. Conf. Elect., Electron. Inf. Eng. (ICEEIE)*, 2021, pp. 1–3.
[40] Y. Huang, L. Shen, and H. Liu, "Grey relational analysis, principal component analysis and forecasting of carbon emissions based on long short-term memory in China," *J. Clean. Prod.*, vol. 209, pp. 415–423, Feb. 2019.
[41] F. Chollet et al., "Keras documentation: Masking and padding with Keras." 2015. [Online]. Available: https://keras.io/guides/
[42] K. Monterrubio-G'omez, N. Constantine-Cooke, and C. A. Vallejos, "A review on competing risks methods for survival analysis," 2022, *arXiv:2212.05157*.
[43] B. Liu, "Cumulative hazard function based efficient multivariate temporal point process learning," 2024, *arXiv:2404.13663*.
[44] S. Sundrani and J. Lu, "Computing the hazard ratios associated with explanatory variables using machine learning models of survival data," *JCO Clin. Cancer Inform.*, vol. 5, pp. 364–378, Mar. 2021.
[45] X. Zhu, J. Yao, and J. Huang, "Deep convolutional neural network for survival analysis with pathological images," in *Proc. IEEE Int. Conf. Bioinf. Biomedicine (BIBM)*, 2016, pp. 544–547.
[46] H. Kvamme, O. Borgan, and I. Scheel, "Time-to-event prediction with neural networks and Cox regression," *J. Mach. Learn. Res.*, vol. 2, no. 1, pp. 1–30, 2019.
[47] H. Steck, B. Krishnapuram, C. Dehing-Oberije, P. Lambin, and V. C. Raykar, "On ranking in survival analysis: Bounds on the concordance index," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 1209–1216.
[48] H. A. Ammar, R. Adve, S. Shahbazpanahi, G. Boudreau, and K. V. Srinivas, "POMDP-based handoffs for user-centric cell-free MIMO networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, 2022, pp. 341–346.

JAVANE ROSTAMPOOR (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from the Iran University of Science and Technology, Tehran, Iran, in 2012 and 2015, respectively, and the Ph.D. degree in electrical and computer engineering focusing on caching optimization in wireless networks from the University of Toronto in 2024, where she is currently a Postdoctoral Fellow working on quantum-enhanced sensors. From June 2015 to December 2018, she worked as a Researcher with the Communication Technology Institute, Iran Telecommunication Research Center, Tehran. In 2023, she had an internship with Ericsson Canada under the Mitacs Accelerate Program. Her current research interests include caching, optimization techniques, signal processing, time-series analysis, dynamic programming, machine learning, and quantum sensing. She is a recipient of the University of Toronto Fellowship, the Nortel Institute Graduate Scholarship for Telecommunications, the Edward S. Rogers Sr. Graduate Scholarship, the H. W. Price Research Fellowship in Electrical Engineering, and the Ewing Rae Graduate Scholarship.

RAVIRAJ S. ADVE (Fellow, IEEE) was born in Mumbai, India. He received the B.Tech. degree in electrical engineering from IT Bombay in 1990 and the Ph.D. degree from Syracuse University in 1996. From 1997 to 2000. he was a Research Associate with Defense Conversion Inc., on contract with the Air Force Research Laboratory, Rome, NY, USA. He joined the Faculty of the University of Toronto in 2000, where he is currently a Professor. His research interests include analysis and design techniques for cooperative and heterogeneous networks, energy harvesting networks, and in signal processing techniques for radar and sonar systems. He received the 2009 Fred Nathanson Young Radar Engineer of the Year Award. His thesis was the recipient of the Syracuse University Outstanding Dissertation Award.

ALI AFANA received the M.Sc. degree in communications engineering from Birmingham University, U.K., in 2009, and the Ph.D. degree in electrical and computer engineering from Concordia University, Montreal, Canada, in 2014. He is a Research and Development 5G/6G Wireless Systems Developer with Ericsson, Canada, where he leads/co-leads the industry-university research collaborations in the areas of next generation radio access networks (RAN) intelligence, low latency communications, and spectrum sharing technologies, drives early-phase system designs and algorithms development, and contributes to intellectual property development (patents) for next generation RANs. Prior to joining Ericsson, he was with Lakehead University and the Memorial University as an Instructor/Postdoctoral Fellow. His research interests include 5G/6G wireless networking, signal processing for communications, and robust machine learning for networks. He is a co-recipient of the IEEE ICC 2022 Best Paper Award.

YAHIA A. ELDEMERDASH AHMED received the B.Sc. and M.Sc. degrees in electrical and computer engineering from Al-Azhar University, Cairo, Egypt, in 2002 and 2007, respectively, and the Ph.D. degree in electrical and computer engineering from Memorial University, St. John's, NL, Canada, in 2015, where he was a Postdoctoral Fellow from 2016 to 2018. From 2003 to 2010, he was a Research and Teaching Assistant with the National Telecommunication Institute, Cairo. He is currently a 5G System Developer with Ericsson, Ottawa, ON, Canada. His research interests include wireless communications and signal processing. He has served as a Technical Program Committee Member for numerous flagship international communication conferences, such as IEEE ICC, VTC, and CrownCom.