STRCACHEML: A MACHINE LEARNING-ASSISTED CONTENT CACHING POLICY FOR STREAMING SERVICES

Arpan Mahara, Jose Fuentes, Christian Poellabauer, Naphtali D. Rishe

Knight Foundation School of Computing and Information Sciences
Florida International University, Florida, USA

ABSTRACT

Content caching is vital for enhancing web server efficiency and reducing network congestion, particularly in platforms predicting user actions. Despite many studies conducted toimprove cache replacement strategies, there remains space for improvement. This paper introduces STRCacheML, a Machine Learning (ML) assisted Content Caching Policy. STRCacheML leverages available attributes within a platform to make intelligent cache replacement decisions offline. We have tested various Machine Learning and Deep Learning algorithms to adapt the one with the highest accuracy; we have integrated that algorithm into our cache replacement policy. This selected ML algorithm was employed to estimate the likelihood of cache objects being requested again, an essential factor in cache eviction scenarios. The IMDb dataset, constituting numerous videos with corresponding attributes, was utilized to conduct our experiment. The experimental section highlights our model's efficacy, presenting comparative results compared to the established approaches based on raw cache hits and cache hit rates.

KEYWORDS

Cache Hit, Cache Miss, Content Caching, Machine Learning (ML), Simulation.

1. INTRODUCTION

The rapidly evolving computing landscape has led to a significant efficiency mismatch between processors and input/output (I/O) peripherals like hard drives, printers, and keyboards. This gap has induced demand for advanced I/O architectures and efficient storage solutions to enhance communication between CPUs and storage devices. This demand is high for a wide range of applications, including real-time applications, gaming, high-performance computing, web services, and, notably, streaming services. These services require a highly efficient cache management mechanism to prevent performance bottlenecks due to frequent read-and-write calls, an operational characteristic that is quite prevalent in streaming services.

Video streaming services operate in a realm where many users frequently and concurrently access vast amounts of data. In such platforms, content caching serves as temporary data storage and has become integral. This strategy involves holding copies of content near where it is frequently requested, increasing data retrieval performance by reducing data access latency. Fast and proficient storage mechanisms, such as Random Access Memory (RAM) and cache memory, are essential to counteract the complexities brought about by these stringent prerequisites. RAM acts as a transient principal memory during software execution, but the data it holds gets lost once the device is powered off. On the other hand, cache memory, being a small and high-speed memory segment, stores data that is accessed often, reducing the frequent need to access the hard disk.

David C. Wyld et al. (Eds): AIBD, MLSC, ACSTY, NATP, CCCIoT, SVC, SOFE, ITCSS -2024
pp. 111-124, 2024. CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.140409

Efficient cache memory management is paramount due to cache memory's limited size and the performance penalty associated with cache misses. A *cache hit* is when the requested data is available in the cache memory, assisting fast data retrieval. In contrast, a *cache miss* arises when the system needs to identify and reallocate data by replacing existing stored information, a comparatively slower process. Various cache replacement strategies strive to uphold a high cache hit ratio, aiming to diminish the frequency of data replacements to the lowest possible extent. Therefore, thorough planning and design are necessary to ensure high-efficiency performance in devices equipped with cache memories.

Though efficient, established cache replacement policies, such as Most-Recently-Used (MRU), LRU, and LFU, are not universally suitable due to their workload dependency and rigid design [9]. Machine Learning (ML) and Deep Learning have proven to be versatile algorithms yielding promising results in various domains, from in-depth biomedical data analysis [4] to financial forecasting [5] and even complex tasks such as satellite image classification [19]. Given their wide range of successful applications, ML and Deep Learning have also been employed in the cache replacement domain to design proficient and effective policies. By leveraging prior data, they can discover patterns hidden in workloads, leading to improved decision-making and potentially a higher cache hit ratio. The rise of powerful computing devices, such as GPUs [22] and Tensor Processor Units (TPUs) [11], has enabled better training and execution rates for ML algorithms.

In this study, we present STRCacheML, an innovative, ML-driven cache replacement policy designed specifically to increase the performance of content caching. STRCacheML harnesses the power of Machine Learning to learn dynamicallyfrom access patterns,improving upon established replacement policies. Our approach aims to enhance the performance of content caching, specifically in the context of streaming services, underlining how a well-calibrated application of Machine Learning can optimize and transform cache management efficiency. Our contributions presented here are:

- We have developed a method for feature construction that takes full advantage of the available parameters on the platform to construct feature vectors for Machine Learning models.
- We have evaluated a variety of Machine Learning and deep learning methodologies on initially constructed features and have selected the most effective model for our application.
- We propose and demonstrate STRCacheML, a new cache replacement policy. By integrating the selected Machine Learning model, STRCacheML makes intelligent cache eviction decisions, enhancing the overall performance of content caching in streaming services.

While ML-based policies promise improved performance, they also introduce new concerns, including the computational complexity associated with the runtime of ML algorithms, the need for training data, and, sometimes, the lack of interpretability of their decisions. Addressing these challenges requires a balanced approach that optimizes cache management efficiency and computational costs. To this end, our work focuses on the application of computationally less expensive ML models such as Random Forest (RF), Decision Tree, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP). While advanced deep learning models can offer powerful predictive capabilities, they also typically require more computational resources and larger datasets for effective training [14], which may not be feasible or necessary in all application scenarios.

2. RELATED WORK

The evolution of cache replacement policies has spanned from standard methods like LRU, LFU, MRU, etc., to advanced methodologies incorporating Machine Learning, Deep Learning, and

Reinforcement Learning (RL) models. Our work, STRCacheML, builds upon these developments to introduce an ML-guided content caching policy.

Established cache management strategies like LRU, MRU, and LFU form the foundation of cache management methodologies, primarily based on recency and frequency of data requests. There has been substantial exploration of these methods in literature. Simultaneously, attempts to apply Machine Learning techniques to augment these strategies have gained momentum. For instance, [3] enhanced the LRU policy using supervised ML Algorithms like SVM, naive Bayes Classifier, and decision tree. Their work involved training ML models to predict data reusability, effectively boosting the LRU policy's performance. Popularity distribution has been a significant factor in cache management. Empirical-theoretical findings in web caching have shaped the understanding of content distribution based on popularity, essentially, the likelihood of being requested in cache memory [8]. This research highlights the empirical data related to the distribution probabilities of popularity events or demands, which are critical to effectively managing web caches. One such practical law, Zipf's, initially proposed for word frequency distribution in a language, states that the n-th most popular item arises with a probability proportional to $1/n^{\alpha}$, where $\alpha > 1$ [34]. In our work, STRCacheML, we leverage these empirical laws, specifically Zipf's law, and synthesize datasets for training and testing our model.

Deep learning techniques have been increasingly applied in cache management domains. For example, Zhong et al. proposed an LSTM-based model designed to predict the properties of objects to be stored in cache memory, which led to enhanced cache performance [33]. Following a similar path, [25] also proposed an LSTM model named Glider to minimize the cache miss rate.

Moving beyond just predictive models, some researchers have combined various cache replacement policies using ML techniques. One such example is [29], which utilized ML to design a hybrid cache replacement method, effectively merging the advantages of both LFU and LRU methodologies. Furthermore, RL algorithms for cache replacement have been explored in recent literature. These techniques operate on the principle that an agent executes various actions within a particular environment to refine its behavior for a specific task. [33] employed deep RL to improve cache performance for web pages, creating a popularity-agnostic model that does not require any information about popularity distribution. Despite these advancements, limitations exist, including high training and execution costs, and the need for more consideration of temporal aspects has been identified [15].

A significant concern is that these advanced approaches are designed for general-purpose caching rather than specifically for content caching. They require substantial computation and need to guarantee effective handling of user preference or performance in content caching platforms. One advanced approach, Raven targets both in-memory and content caching [12]. It employs a Mixture Density Network to learn the patterns of objects' next-request arrival times and uses a Gated-Recurrent Neural Network (GRU) to understand temporal dependencies in the data [12]. However, applying GRUs in the content caching platform can increase computational costs due to their complexity and potential for gradient bursting during loss calculations [21].

In 2021, a content caching method was proposed for video streaming in a cloud-edge cooperative platform [17]. This paper's strategy includes two main steps: clustering edge servers using the k-means algorithm and analyzing latency and caching costs to determine optimal content caching policies. Though innovative, this strategy is edge-cloud-oriented and may not effectively handle user preferences in streaming platforms like Netflix, Prime Videos, and so on, due to itsdesign [17].Hence, to address the concerns mentioned above, we propose STRCacheML, a smart content caching strategy suitable for streaming platforms that can handle users' preferences with the help of a ML model(for a high-level overview of the workflow, refer to Figure 1).

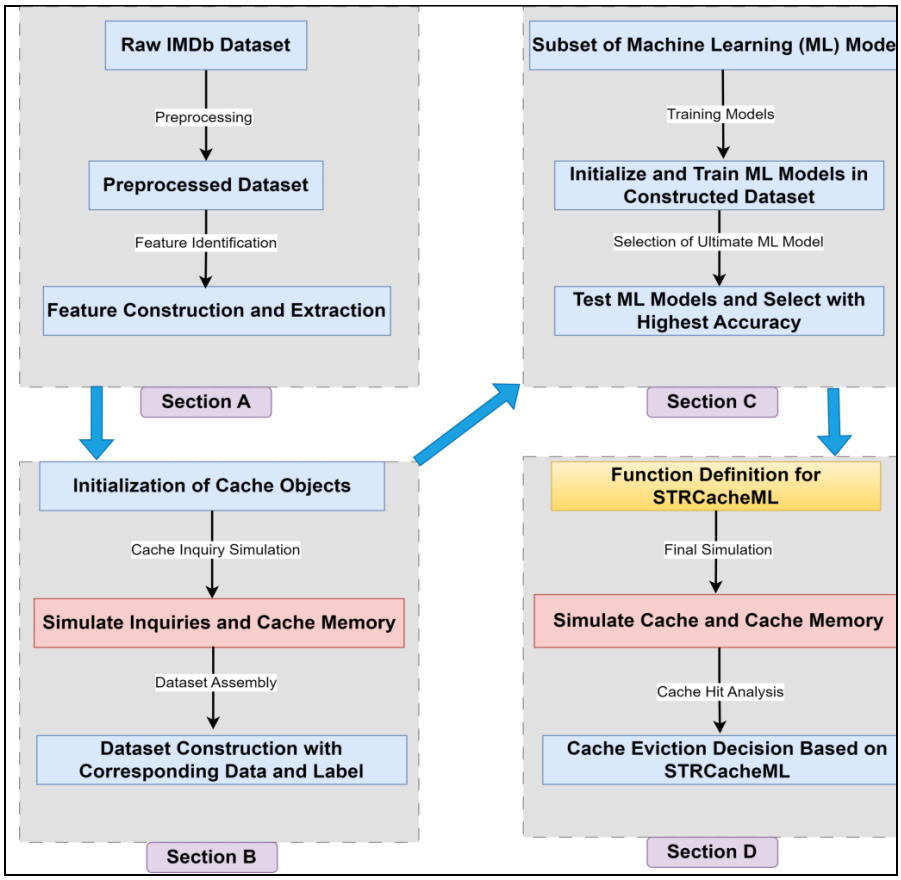

Figure 1. Workflow Diagram of the STRCacheML Process.

3. PROPOSED APPROACH

We propose "STRCacheML", an innovative content caching replacement policy guided by a Machine Learning (ML) algorithm. Our primary objective is to enhance content caching in streaming services by capitalizing on available attributes to construct feature vectors. The constructed feature vectors guide our ML model during the training and testing phases to make intelligent cache replacement decisions.

While STRCacheML is designed to operate in a broad range of environments, it is worth noting that specific adaptations may be necessary to suit different datasets or settings. For instance, during our experiment with the IMDb dataset, we encountered a lack of real inquiry data. To overcome this, we introduced a simulation using a probability distribution to mimic a real-world environment. A detailed explanation of this adaptation and our experimental setup will be provided in a subsequent section.

Our study examined a range of accessible and computationally efficient Machine Learning algorithms, including the Multilayer Perceptron (MLP) from the deep learning domain. This approach was motivated by our aim to explore and exploit the computational and performance characteristics of these models, which are both accessible and less computationally intensive, within a content caching platform.
The components of our approach are listed in the following subsections:

3.1. Feature Vector Construction

We consider $O = \{o_1, o_2, o_3, \ldots, o_n\}$, the set of all objects available on the platform. We represent all the objects, $o \in O$, by a feature vector, $F(o)$, including attributes such as one-hot encoding of genres, the object's popularity over a specific period, historical inquiry data, object's size, etc. We represent each object by a feature vector, $F(o)$, which can be mathematically illustrated as follows:
For every object $o$in the set of all objects O, let $A = \{a_1, a_2, a_3, \ldots, a_m\}$be the set of all possible attributes. The features vector is built as,

$$F(o) = a_1 \oplus a_2 \oplus \cdots \oplus a_m \quad (1)$$

where each a_i represents an attribute of o written as a vector and $\oplus$corresponds to vector concatenation.

3.2. Cache Memory and Inquiry Simulation

Let Crepresent our cache with a limited size m, where each content, $c \in C$, is an object o from our set O. We simulate object requests using a probability distribution, $P(o)$, defined on the set$O$ where the probability of being requested is proportional to the popularity of the given object o (See Figure 2). The simulation is modeled as follows: For all contents, $c \in C$, where $C \subseteq O$, draw samples o to be processed on the cache C by "STRCacheML".
Moreover, the cache has its probability distribution defined for the objects $c \in C$, which is kept tracked during the execution let r_c denote the number of inquiries made to object c until the current date. The total number of inquiries for all objects is represented as $\sum_{o \in C} r_o$ and the probability distribution of requests is given by

$$P_C(c) = \frac{r_c}{\sum_{o \in C} r_o} \quad (2)$$

3.3. Training and Selection of ML Algorithm

A series of ML models, $\{M_1, M_2, \ldots, M_k\}$, (where k denotes the total number of ML algorithms selected) are trained on these feature vectors, $F(o)$, obtained from each $o \in O$. After training, we evaluate the models' performance using both the training and testing datasets, focusing on the accuracy metric. With this assessment, we identify the model that provides the highest accuracy, which will be selected as our preferred ML model, $M^{\star}$. This procedure can be illustrated as follows:
A set of ML models, $\{M_1, M_2, \ldots, M_k\}$are trained using the feature vector$F(o)$, as constructed in equation (1). Among all trained models, we identify the optimal ML model as $M^{\star}$for our cache replacement policy. The selection criterion is given by the model's accuracy, as formalized in the following equation:

$$M^{\star} = \text{argmax}_{M_i \in \{M_1, \ldots, M_k\}} \; accuracy(M_i) \quad (3)$$

3.4. Cache Eviction

When cache C reaches its capacity (i.e., $|C| = m$) and a cache miss occurs, the ML model, $M^{\star}$, is employed to determine the likelihood, $L(o)$, of each object o in cache C being requested again. The object with the least likelihood, $L(o)$, as computed by $M^{\star}(F(o))$, is selected for eviction to make space for new content in the cache.
Given a cache miss M and the cache memory is full, i.e., $|C| = m$, the likelihood of each object being requested again is computed as$L(o) = M^{\star}(F(o))$for each $o \in C$. Finally, the object $\hat{o}$to be evicted is expressed as:

$$\hat{o} = \text{argmin}_{o \in C} \, L(o) \quad (4)$$

3.5. STRCacheML

By integrating equations (1), (2), (3), and (4), we formulate STRCacheML, our proposed cache replacement policy. This policy can be outlined as follows:
Given an object inquiry (request) for any $o \in O$:

1) If $o \in C$(i.e., a cache hit), the inquired object is acquired from the cache without initiating the cache replacement procedure.
2) If $o \notin C$(i.e., a cache miss) and $|C| < m$, the object is first fetched from the backend (main memory in the platform) and added to the cache.
3) If $o \notin C$(i.e., a cache miss) and $|C| = m$(i.e., the cache is full), we calculate the likelihood $L(o')$of each object $o' \in C$in the cache being requested again, as defined in equation (4) above. The object $\hat{o} = \text{argmin}_{o' \in C} L(o')$is evicted from the cache, and the requested object o is accessed from the main memory and added to the cache.

Through these steps, we introduce STRCacheML, an ML-based cache replacement policy that leverages rich feature vectors and predictive modeling to make intelligent content caching decisions, potentially enhancing the cache replacement performance in streaming services.

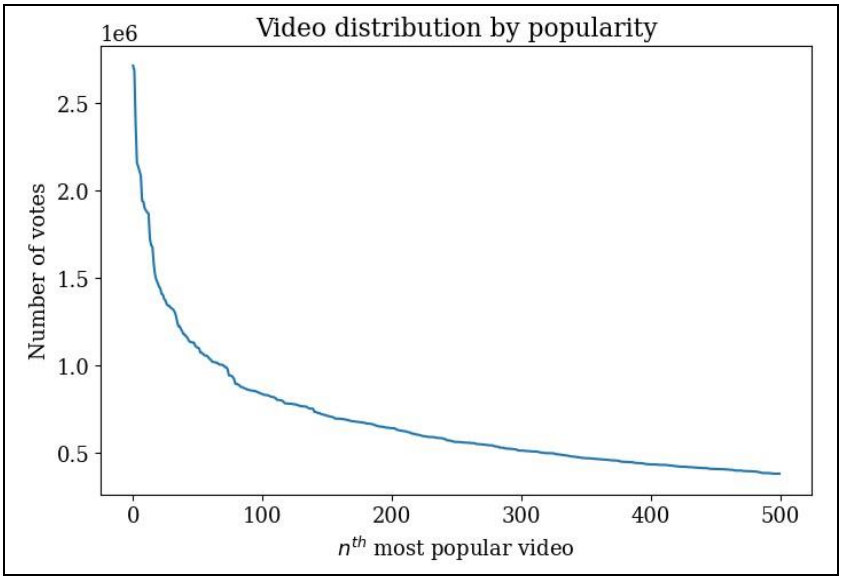

Figure 2. Video popularity distribution for the 500 most popular videos in IMDb. It is interesting how video distribution follows Zipf's law if they are sorted by some popularity attribute.

4. DATA PREPROCESSING, SIMULATION AND DATASET PREPARATION

Data preprocessing, a critical aspect of the Knowledge Discovery from Data (KDD) process, involves a set of techniques employed before applying data mining methods. This technique handles inconsistencies, redundancies, and other data imperfections, making the data suitable for the chosen Machine Learning algorithm.

In our study, we used a dataset sourced from IMDb [13] for experimental purposes, potentially showcasing the efficiency of our policy in a video streaming platform. Since the data obtained from IMDb were raw and unprocessed, preprocessing steps were required to prepare the dataset for Machine Learning applications. First, we utilized Python libraries such as pandas and NumPy to handle null values, mostly deleting the data entries from the dataset. To remove discrepancies in the dataset, we used the Scikit-learn library, specifically its preprocessing module, to maximize

the normalization of the data as needed. Lastly, we decided to disregard specific attributes that were deemed unsuitable for training our ML models, focusing on five attributes:"Primary Title","Start Year","Run Time","Genres","Average Rating," and "Number of Votes". In the absence of direct information on video sizes within our dataset, we adapted the "Run Time" attribute availableto act as a proxy. To the best of our knowledge, we believe this adaptation presents a practical approximation for cache replacement policies often influenced by video sizes.Therefore, even within the simplification and simulation, our model maintains a level of accuracy to a certain valuable extent in representing the characteristics influential to real-world caching scenarios.

We created a virtual cache with predetermined memory limits to simulate a realistic environment for our cache replacement policy, STRCacheML.In the absence of real-world inquiry data, we generated a series of 10000 cache inquiries based on a probability distribution derived from the "Number of Votes" each video received on IMDb. These votes were taken as a proxy for the videos' popularity, whichsuggested their likelihood of being requested (refer to Figure 2, which illustrates the probability distribution across the videos). The outcome of each simulated inquiry was determined by whether the requested video was present in our cache, resulting in either a cache hit or miss.

From the initial simulation of inquiries using the IMDb dataset, we constructed a comprehensive dataset comprising 19992 samples. Each sample was associated with a feature vector, denoted as $F(o)$, representing the attributes of the requested video,including genre, run time, and calculated popularity. These feature vectors were then labeled with binary values, assigning "1" for a cache hit and "0" for a cache miss. This labeling process was important for training the Machine Learning models, enabling them to understand the feature vectors that contribute to cache hits or misses. Such understanding is contributory in producing likelihood estimations that inform cache eviction decisions in subsequent phases.Thus, our dataset comprised these feature vectors $X = [x_1, x_2, \ldots, x_n]$along with their corresponding binary labels, forming the foundation for training our Machine Learning models to learn from historical cache performance under conventional policies and make predictive decisions for future cache management.

By integrating both staticattributes of videos and dynamic user interaction data in a simulated environment, our dataset provides Machine Learning models with a comprehensive understanding of video demand over time. By incorporating historical usage data along with evolving trends in video popularity within a simulated environment, our methodology ensures that STRCacheML adapts to changing user behaviors,enhancing the predictive accuracy and practical applicability of our cache replacement policy.

4.1. Selection of ML Algorithms and Hyperparameter Tuning

Building upon the earlier discussion, feature selection and hyperparameter tuning play an instrumental role in achieving accurate models [26]. The training parameters, model architecture, and features can significantly determine model performance. During the selection of ML algorithms, observing each algorithm's performance on both the test and training sets is crucial to recognize if the model is overfitting, which memorizes training features rather than learning the underlying patterns. Overfitting can lead to increased computational complexity, reduced accuracy, and false confidence in the model's predictions [32].

In our study, we conducted experiments divided into two phases. First, we trained different machine learning models as described in building a query dataset based on the probability of being requested $P(o)$ where the object to be queried next is predicted. For each model,80% of the dataset was used as a training set, whereas the remaining 20% was used to test the model's

capabilities.It is worth mentioning that this dataset setupprovides a platform for training and evaluating supervised ML algorithms.Second, we choose the best ML usingequation (3) tobuild the cache replacement policy.We initially selected the Random Forest (RF) algorithm to understand its potential usability in our mechanism. Random Forest, developed by Leo Breiman and Adele Cutler, is a widely adopted ML algorithm. It traverses multiple decision trees, each contributing to sub-decisions and aggregating their outcomes to make a final decision [35]. Due to its effectiveness in classification and regression problems, we successfully tested RF within our dataset's classification setup. Initially, we set the maximum number of features equal to the number of extracted features, and the maximum depth was set to 70. This setting means that the same number of features will be selected in each split, and the tree in the model will have, at most, 70 levels of nodes. However, after initial training and testing, the maximum number of features was adjusted to the square root of the extracted features, with the tree depth remaining the same, to fine-tune the process and prevent overfitting.

Similarly, we adapted the Gradient Boosting Machine (GBM), a widely recognized ML algorithm proficient inclassification and regression tasks. GBM operates by iteratively refining decisions through ensemble models, typically decision trees, and focuses on correcting the errors of previous iterations [36]. Given its successful application in areas such as web search engines, we decided to implement the GBM model available in the sklearn library. In our experimentation with various parameter settings, we observed minimal variations in accuracy for our binary classification dataset. Consequently, we configured GBM with 100 estimators, a learning rate of0.1, anda maximum depth of 2, and we recorded both training and testing accuracies to compare with other models. Next, the SVM algorithm was trained on the model, setting the regularization parameter "*C*" to 100 and the gamma parameter for the Radial-Basis-Function (RBF) kernel to "auto", which means that it is computed from the data. Parameter "*C*" maintains a trade-off between maximizing the decision boundary and minimizing the classification error, while the gamma parameter helps to measure the similarity between two data points [7]. Likewise, the K-Nearest Neighbors (KNN) algorithm was trained on the dataset, initially considering three nearest neighbors while predicting new values. For fine-tuning, the model was adjusted to consider the five nearest neighbors contributing to the prediction with the inverse of their respective distances from the query point.

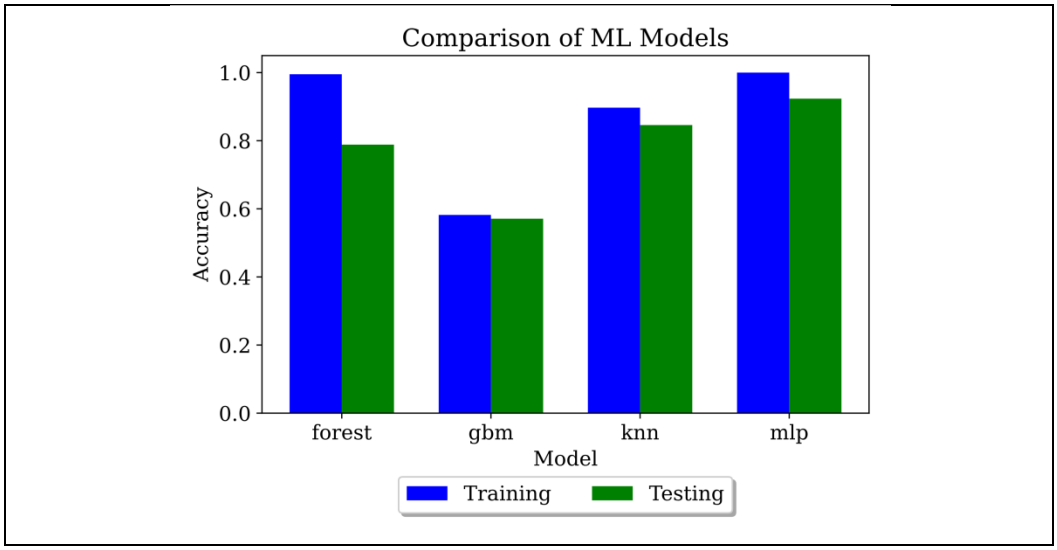

Figure 3. Comparison of Training and Testing Accuracies Across All Evaluated Machine Learning Models

Ultimately, we selected the Multi-Layer Perceptron (MLP) as our final ML model for experimentalpurposes, assessing itspotential for integration into our cache management policy. Initially, the MLP model was designed with two hidden layers comprising 128 and 64 nodes, respectively. To enhance the model's performance, we adopted an iterative approach using Scikit-Learn's GridSearchCV for automated hyperparameter tuning following the idea mentioned in [2]. The architecture of the hidden layers was systematically varied across multiple training iterations, altering the number of layers and nodes within each layer.However, during this process, we discovered a critical insight. Augmenting the number of layers did not necessarily enhance the accuracy of our model. Instead, it only increased the time complexity of the model while the accuracy plateaued. GridSearchCV allowed us to navigate through multiple combinations of parameters and revealed that a model with two hidden layers of 256 and 128 nodes, respectively, provided an optimal balance between accuracy and computational efficiency for our dataset. Consequently, we decided to adhere to this model structure. Upon comparison of accuracy and time complexities among above mentioned ML models, the MLP model concluded as the most efficient with an impressive testing accuracy of 0.97 on the dataset, surpassing the accuracy levels of SVM, GBM (around 0.60), and RF (around 0.84) and KNN (around 0.84)as shown in Figure 3. Hence, MLP was chosen as the final ML model for our cache replacement policy.

5. SIMULATIONS, COMPARISONS, AND EVALUATION

We conducted our experiments ina Python 3.10.12 environment, utilizing the TensorFlow frameworkfor machine learning computations. The computational workloadwas managed by an NVIDIA T4 GPU with15GB of GPU RAM. Throughout our study, we conduct experiments using a fundamental cache structure, divided into individual slots capable of storing one video each. Our reference for cache capacity is its *cache size*, the number of slots. We evaluate the performance of our proposed model, STRCacheML, by comparing it with established caching policies like LRU, LFU, and Least Recently Frequently Used (LRFU). We utilize raw cache hit and cache hit rate as our key computational and comparison metrics. The raw cache hit measures the instances where the requested object is available in the cache. On the other hand, the cache hit rate provides a more insightful metric, indicating the proportion of total requests resulting in a cache hit. It is derived from the formula:

$$\text{Cache Hit Rate} = \frac{\text{Cache Hits}}{\text{Cache Hits} + \text{Cache Misses}} \tag{5}$$

For STRCacheML, we implement a predictive and adaptive approach assisted by a trained MLP model, which sequentially updates based on inquiries and cache hits during the simulations. The MLP model, trained on historical video access data, predicts the popularity trend of videos, aiding dynamic cache management. When a video request occurs and the video is absent in the cache, STRCacheML utilizes its core predictive principle and assigns each video in the cache a score corresponding to its predicted future popularity. During the eviction procedure, the video least likely to be requested again is removed from the cache to accommodate new content. This approach inherently considers user preferences, as it is driven by user behavior and inquiry patterns. Therefore, it allows STRCacheML to adapt to changing user preferences, an important aspect in content caching.

We carefully applied STRCacheMLalong with the established cache policies to the final phase of simulated cache inquires and virtual cache memory to accurately measure the cache hits and cache hit rates. For established caching policies like LRU, LFU, and LRFU, we thoroughly simulated their principle on the dataset. For LRU, we keep track of the timestamp for each video, reflecting the time an inquiry was made for the given video. Also, for LFU, we applied its principles by computing a frequency count for each video in the cache, recording the total number of requests made, and evicting the video based on frequency. For LRFU, we designed a

function to dynamically update each video's Combined Recency and Frequency (CRF) value upon each inquiry. Eviction from the cache was based on the CRF values calculated for each video.

We conducted experiments on two different simulations, Simulation1 and Simulation2, designed to represent different usage scenarios. In Simulation1, we constructed a scheme that involved 10000 video queries and a cache size of 25 slots. This scenario was chosen to simulate a lower to moderate level of user demand, where the cache size and number of queries were small compared to Simulation2. The selected cache replacement policies, including LRU, LFU, LRFU, and STRCacheML, were sequentially implemented and the count of cache hits along with the cache hit rates were recorded

For the second simulation, Simulation2, we escalated the volume of video queries and cache size to 20000 queries and 50 slots, respectively. This simulation was designed to represent a situation of high demand, with significantly more queries and a larger cache size than the first simulation. Just as in Simulation1, the same cache replacement strategies were utilized, the record of cache hits was noted, and the cache hit rate was consequently computed.

By observing and comparing the results of both simulations, we gained valuable insights into the performance of different cache replacement policies under two different conditions. Table I illustrates how our proposed STRCacheML model performed well over other strategies in both simulations when assessed based on raw cache hits and cache hit rates. The table delivers a comparison among the cache replacement policies, presenting the number of cache hits and the corresponding cache hit rates accomplished in both simulations. In addition, the cache hit rates of each model were visualized in Figures 4 and 5, which provide a more detailed comparison of each model's performance. These results highlight the better performance of our proposed STRCacheML model in different scenarios.

TABLE I. CACHE HITS AND HIT RATES FOR DIFFERENT MODELS IN SIMULATION1 AND SIMULATION2

Model	**Simulation1**		**Simulation2**	
	Raw Cache Hits	**Hit Rate (%)**	**Raw Cache Hits**	**Hit Rate (%)**
LRU	644	6.44	2484	12.42
LFU	675	6.75	2865	14.33
LRFU	758	7.58	3422	17.11
STRCacheML	1078	10.78	3770	18.85

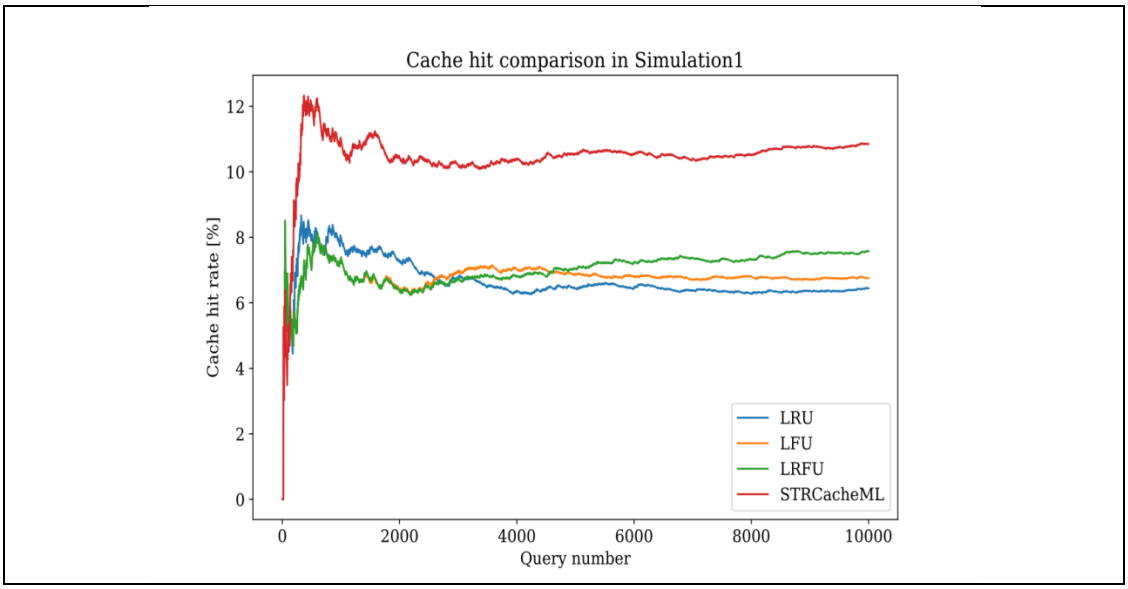

Figure4.Comparison of Cache Hit Rates Over 10,000 Simulated Inquiries

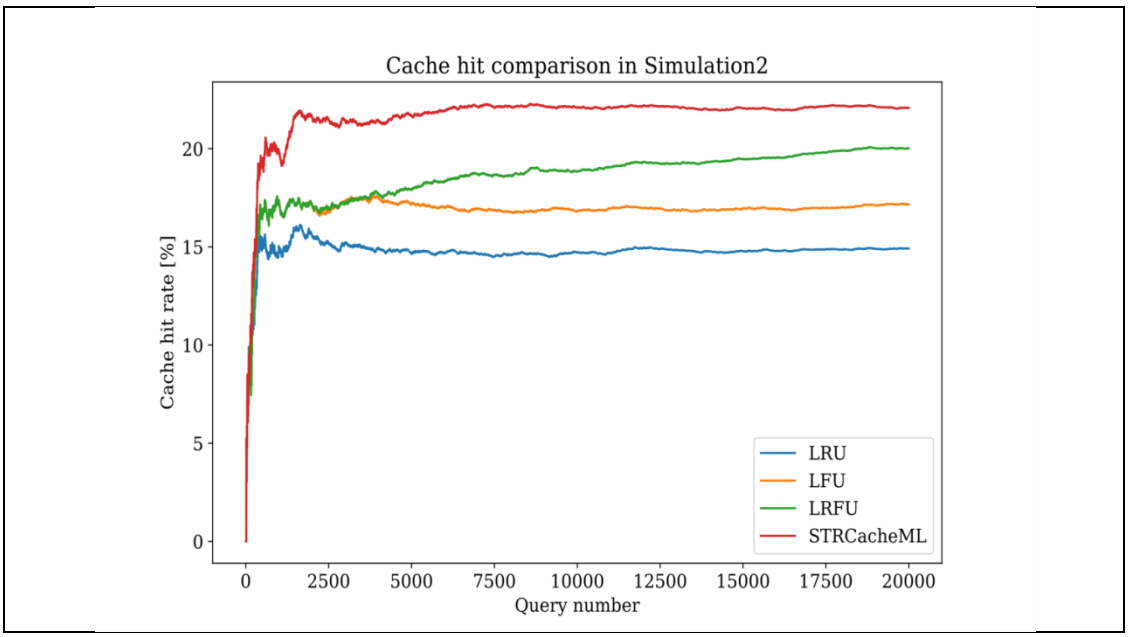

Figure5.Comparison of Cache Hit Rates Over 20,000 Simulated Inquiries

6. CONCLUSIONS AND FUTURE WORK

In this study, we presented STRCacheML, an innovative ML-guided cache replacement policy to enhance content caching in streaming services. Through experiments and simulations of real-world scenarios, STRCacheML demonstrated better performance over established caching policies such as LRU, LFU, and LRFU. A significant factor contributing to this enhanced performance is STRCacheML's ability to consider user preferences and adapt to evolving popularity trends, a critical aspect in efficient content caching. By dynamically predicting and managing the popularity trend of videos, STRCacheML improves cache hit rates. Specifically,

STRCacheML achieved an increase of approximately 5%, 4%, and 3% in cache hit rate over LRU, LFU, and LRFU, respectively, in Simulation1, as shown in Figure4. In Simulation2, STRCacheML also exhibited a consistent enhancement in cache hit rates in comparison to other models, as shown in Figure5. An interesting observation was that each model demonstrated a sequential improvement in cache hits, validating their respective principles within a content caching platform.However, integrating assisted cache policies introduces additional computational overhead compared to their classical counterparts, which require minimal information and straightforward implementation. The extent of this added overhead is contingent on the size of the ML model, determined by the feature vector size and hyperparameters. With fixed hyperparameters, strategic feature engineering or a projection algorithm mitigates high computation overhead, maintaining a "positive" trade-off where the gain in cache hit ratio outweighs the produced overhead. We don't address overhead during the training process, as this computationally intensive phase can be conducted offline without impacting the user's experience.

Streaming platforms handle large datasets. To design ML-assisted scalable cache policies, integrating feature engineering techniques, as demonstrated in this work, is essential for lightweight ML algorithms suitable for online execution. Successful implementation demands properly integrating information obtained from clients engaging with diverse streaming services. If integrated effectively, STRCacheMLcan decrease traffic congestion, as the requested data could be efficiently fetched from cache memory. This approach could reduce energy consumption and prevent overloading through improved data handling.Expanding beyond streaming, this approach proves beneficial in various domains such as e-commerce, content delivery networks (CDNs), and social media platforms. Thisis because our approach’s scalability relies on available object parameters, a common aspect across different content delivery systems. In these contexts, a cohesive interplay between user-generated data and the design, training, and development of ML cache policies becomes helpful for improving content delivery and user experience. Another notable concern ariseswith user data privacy. It is important to note that approaches relying on training from user data involve learning patterns from potentially sensitive information. Various established techniques, including anonymization and more robust cryptographic methods, such as differential privacy or direct data encoding, can be utilized to address this issue. In the latter case, methods like Multi-Party Computation can be integrated to enable computation on encrypted data, ensuring that the data remains private. These alternatives present diverse options for tackling the challenge of preserving privacy in our methodology, STRCacheML,whileextracting learnablepatternsfrom user data.

In conclusion, STRCacheML, by integrating Machine Learning techniques with content caching, demonstrates an advancement in the domain of Machine Learning Applications, particularly in information retrieval and intelligent cache management systems, leading to improved efficiency and adaptability in streaming services. With its potential to extend benefits to CDN platforms beyond content caching in streaming services, STRCacheMLmay encounter some limitations within specific scopes. These limitations can be addressed through broader experimentation and dataset expansion, areas we identify as opportunities for future research.
Future work will explore more advanced ML and Deep Learning techniques, such as Recurrent Neural Networks and Transformers, to be integrated into STRCacheML, also maintaining computational efficiency and user privacy. There could be a potential benefit of incorporating a hybrid method into the cache replacement policy. Additionally, investigating other possible attributes and sophisticated feature engineering strategies by generalizing the applicability could enhance model performance while maintaining computational efficiency.

ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under Grant Nos. CNS-
2018611 and CNS-1920182, FDEP grant C-2104, and DHSgrant E2055778.

REFERENCES

[1] Mart´ın Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *Osdi*, volume 16, pages 265–283. Savannah, GA, USA, 2016.

[2] Tanay Agrawal and Tanay Agrawal. Hyperparameter optimization using scikit-learn. *Hyperparameter optimization in machine learning: make your machine learning and deep learning models more efficient*, pages 31–51, 2021.

[3] Waleed Ali, Sarina Sulaiman, and Norbahiah Ahmad. Performance improvement of least-recently-used policy in web proxy cache replacement using supervised machine learning. *International Journal of Advances in Soft Computing & Its Applications*, 6(1), 2014.

[4] Pierre Baldi. Deep learning in biomedical data science. *Annual review of biomedical data science*, 1:181–205, 2018.

[5] Alessandro Baldo, Alfredo Cuzzocrea, Edoardo Fadda, and Pablo G Bringas. Financial forecasting via deep-learning and machine-learning tools over two-dimensional objects transformed from time series. In *Hybrid Artificial Intelligent Systems: 16th International Conference, HAIS 2021, Bilbao, Spain, September 22–24, 2021, Proceedings 16*, pages 550–563. Springer, 2021.

[6] Leon Bottou. Large-scale machine learning with stochastic gradient descent. In´ *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[7] Richard G Brereton and Gavin R Lloyd. Support vector machines for classification and regression. *Analyst*, 135(2):230–267, 2010.

[8] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 1, pages 126–134. IEEE, 1999.

[9] Kai Cheng and Yahiko Kambayashi. Lru-sp: a size-adjusted and popularity-aware lru replacement algorithm for web caching. In *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000*, pages 48–53. IEEE, 2000.

[10] Hong-Tai Chou and David J DeWitt. An evaluation of buffer management strategies for relational database systems. *Algorithmica*, 1(1-4):311–336, 1986.

[11] Google Cloud. An in-depth look at google's first tensor processing unit (tpu), 2017.

[12] Xinyue Hu, Eman Ramadan, Wei Ye, Feng Tian, and Zhi-Li Zhang. Raven: belady-guided, predictive (deep) learning for in-memory and content caching.In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, pages 72–90, 2022.

[13] IMDb. Imdb datsasets. https://www.imdb.com/interfaces/, February 2023. Accessed: 2023-02-25.

[14] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021.

[15] Jobin Jose and N Ramasubramanian. Applying machine learning to enhance the cache performance using reuse distance. *Evolutionary Intelligence*, pages 1–22, 2022.

[16] Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323, 1979.

[17] Chunlin Li, Yong Zhang, Mingyang Song, Xin Yan, and Youlong Luo. An optimized content caching strategy for video stream in edge-cloud environment. *Journal of Network and Computer Applications*, 191:103158, 2021.

[18] Tian Luo, Siyuan Ma, Rubao Lee, Xiaodong Zhang, Deng Liu, and Li Zhou. S-cave: Effective ssd caching to improve virtual machine storage performance. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 103–112. IEEE, 2013.

[19] Arpan Mahara and Naphtali Rishe. Integrating location information as geohash codes in convolutional neural network-based satellite image classification. *IPSI Transactions on Internet Research*, 19(2):pp.24–30, 2023.

[20] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*, pages 48–53, 2018.

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[22] Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880, 2009.

[23] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2135–2135, 2016.

[24] Subhash Sethumurugan, Jieming Yin, and John Sartori. Designing a cost-effective cache replacement policy using machine learning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 291–303. IEEE, 2021.

[25] Zhan Shi, Xiangru Huang, Akanksha Jain, and Calvin Lin. Applying deep learning to the cache replacement problem. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 413–425, 2019.

[26] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[27] Debabala Swain, Bijay Paikaray, and Debabrata Swain. Awrp: adaptive weight ranking policy for improving cache performance. *arXiv preprint arXiv:1107.4851*, 2011.

[28] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *Acm computing surveys (csur)*, 53(2):1–33, 2020.

[29] Giuseppe Vietri, Liana V Rodriguez, Wendy A Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. Driving cache replacement with ml-based lecar. In *HotStorage*, pages 928–936, 2018.

[30] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C Steely Jr, and Joel Emer. Ship: Signature-based hit predictor for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 430–441, 2011.

[31] Carole-Jean Wu, Aamer Jaleel, Margaret Martonosi, Simon C Steely Jr, and Joel Emer. Pacman: prefetch-aware cache management for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 442–453, 2011.

[32] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[33] Chen Zhong, M Cenk Gursoy, and Senem Velipasalar. A deep reinforcement learning-based framework for content caching. In *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6. IEEE, 2018.

[34] George Kingsley Zipf. Human behaviour and the principle of least-effort. cambridge ma edn. *Reading: Addison-Wesley*, 24, 1949.

[35] Leo Breiman. "Random forests." *Machine learning* 45 (2001): 5-32.

[36] Jerome H. Friedman. "Greedy function approximation: a gradient boosting machine." *Annals of statistics* (2001): 1189-1232.