Predictive Modelling for Next API Call Sequence in Content Delivery Networks

Galiiabanu Bakirova, Markus Sosnowski *
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: galiiabanu.bakirova@tum.de, sosnowski@net.in.tum.de

***Abstract*—Content Delivery Networks (CDNs) play a crucial role in enhancing user experience by caching frequently accessed content. However, challenges arise when users request uncached content, leading to potential disruptions and impacts on website performance. To address this, researchers advocate for proactive caching, pre-emptively storing predicted future requests. This paper explores the application of a language model to server logs for predicting users' API requests. Training the model on logged API calls, we analyze the potential of language models in gaining insights into user behavior. Despite challenges with dynamic data, the model detects recurring patterns and learns API semantics. While results may vary for sites with dynamic structures, this approach opens avenues for future improvements, such as introducing probability thresholds or focusing on specific API endpoints. Challenges persist, requiring each website to train its own model based on its API structure. Our exploration provides valuable insights into the possibilities and limitations of language model-driven API request prediction.**

***Index Terms*—content delivery networks, api, prediction**

1. Introduction

In our rapidly evolving world, website owners often rely on Content Delivery Networks (CDNs) to speed up loading times for end-users. Yet, this decision poses its own set of challenges, including determining which content to cache on the CDN and minimizing cache misses [1].

CDNs typically store popular and frequently accessed content, a strategy that significantly reduces latency and improves user experience. However, challenges arise when a user requests content that is not readily available in the CDN's cache. In such cases, the CDN must either retrieve the resource from the origin server or return an error message to the user. This can lead to disruptions in the user experience and potentially impact overall website performance [2].

To address this challenge, researchers have proposed the idea of predicting users' next requests and preemptively caching the required resources. In their survey, Nanopoulos et al. [3] advocated for storing such items in the cache prior to an explicit request being made. This approach, known as proactive caching, aims to minimize the latency associated with fetching resources from origin servers, ensuring that users enjoy seamless and responsive online experiences.

Building upon this concept, machine learning models have emerged as promising tools for predicting users' next requests. These models can analyze vast amounts of server logs and historical data to identify patterns and predict user behavior. This information can then be used to preload relevant resources in the CDN cache, further enhancing the user experience and reducing latency. [4]

In this paper, we explore the application of a language model approach to server logs datasets to gain valuable insights into user behavior and predict their next API requests.

2. Background and Related Work

A Content Delivery Networks (CDN) is a network of servers strategically located around the world that help deliver web content to users with improved performance and reduced latency [5]. Websites and online platforms often utilize CDNs to enhance their services and meet the growing demands of users.

CDNs work by storing cached copies of web content, such as images, videos, and static files, on servers distributed across various geographic locations. When a user requests content, the CDN automatically directs the request to the server closest to the user, minimizing the distance the data needs to travel. This proximity helps reduce latency and improves the overall loading speed of web pages [2].

In addition to improving performance, CDNs can also help mitigate traffic spikes and distribute the load on servers, ensuring smooth and uninterrupted access to web content even during periods of high demand. By offloading the delivery of content to a CDN, websites can optimize their infrastructure, enhance user experience, and better handle global traffic [2].

2.1. Web Usage Mining

Colley et al. [6] were some of the first ones interested in predicting user's requests. Knowing user's intentions can create a seamless user experience, increase conversion and sales [7] if the users are recommended something they did not know they needed [8]. In the context of server and CDNs, knowing user intentions can be used to predict the user's next API request and preload the resources. This would decrease loading time for the users and ensure effective user interaction. [3]

There are several proposed web usage mining approaches for working on server logs [9]:

 doi: 10.2313/NET-2024-04-1_07

- Association rules - the technique for finding the web pages visited together. One disadvantage of the associate rules approach is that it does not take into account the notion of time difference.
- Frequent Sequences - considering ordered time-sensitive sequences. This technique tries to discover sequence patterns followed by users.
- Frequent Generalized Sequences - a relaxation of frequent sequences, that allows to study user's navigation in a flexible way. [10]

Gery et al. [9] evaluated the three web usage mining approaches on datasets of different sizes and discovered that the Frequent Sequences (FS) performed best in terms of accuracy. The authors emphasize the suitability of the FS technique for analyzing time series and propose an optimal user session time of 25 minutes.

Nigam et al. [11] tried a different approach to predicting user requests. The research studies the effect of Markov model depth on the user's next request prediction. Nigam et al. compare first-, second- and third-order Markov models for predicting the next web page. They propose metrics such as model generation time, prediction time prediction accuracy and coverage for measuring prediction success. In [11] Nigam et al. perform experiments on three datasets. However, the proposed test datasets only have between 29 and 92 different pages, which would correspond to the number of unique API calls.

2.2. Content Delivery Networks Scaling

Content networking is gaining popularity as a go-to technology because it significantly boosts enterprise network performance for media-rich content, all while keeping costs lower compared to traditional methods of web scaling [12]. As CDNs expand their user base, the content stored on CDNs becomes very diversified. Each content category imposes distinct demands on the CDN's caching systems. CDN has to introduce various configuration parameters in order to be able to serve such a wide range of content [13], [14]. Manual tuning of these parameters can be challenging.

With the pursuit of increased efficiency and reduction of cache misses, some reinforcement learning techniques to autonomously manage resources were proposed [13], [15], [16]. Current approaches to caching predominantly utilize "model-free" reinforcement learning (RL), where the system embarks on the learning process without knowledge of the underlying structure, free from any preconceived notions or biases about the task at hand [17], [18]. These systems learn decision-making through first-hand experience, guided by a reward mechanism that reinforces the right decision-making and encourages continued exploration of effective strategies.

While model-free RL holds immense promise for optimizing caching strategies, the RL community has identified three major hurdles that need to be addressed:

- Data-Intensive Learning: Model-free RL algorithms typically require vast amounts of training data, usually in the millions of samples. Accumulating such a large dataset can be time-consuming and resource-intensive [16].
- Overfitting Vulnerability: Model-free RL algorithms are susceptible to the risk of overfitting to the train data. This means that they may perform well on the training set but struggle to generalize to new and unseen data. In the context of caching, overfitting could lead to suboptimal caching decisions [19].
- Complex Debugging and Maintenance: Model-free RL algorithms can be sensitive to hyperparameters, which make them extremely difficult to debug [15], [20].

These challenges pose significant obstacles to the practical implementation of model-free RL for caching in CDN servers.

2.3. RNNs for API Requests Analysis

Reddy and Rudra [21] applied RNN for detecting injections in API requests. They compare three popular RNN approaches for sequential data analysis: bidirectional Vanilla-RNNs, bidirectional LSTMs and bidirectional gated recurrent units (GRU) for requests classification. The obtained results prove the effectiveness of RNN approach on a API request data. Reddy and Rudra were able to achieve the accuracy of 97% for the bidirectional LSTMs and 98,5% for bidirectional GRUs. Arivukarasi and Antonidoss [22] were also able to exploit natural language processing (NLP) approach with RNNs to detect phishing URLs. They achieved the highest accuracy of 98% using RNNs with LSTM layer.

3. Methodology

We applied the language model approach for training the API request prediction model. In order to train the language model for predicting the next API call, we used a recurrent neural network (RNN). RNNs are designed to work with sequential data and are, therefore, the perfect choice for processing a series of API requests. A model was created and trained with the TensorFlow Keras API. The architecture consists of three layers: Embedding layer, Long Short-Term Memory (LSTM) layer, and dense layer. The Long Short-Term Memory layer is the key to capturing long-term dependencies and is suitable for predicting the next API calls based on several previous calls. The architecture of the RNN used is depicted in Figure 1.

The embedding layer is responsible for converting the integer-encoded token (in our case, an API request) into a dense vector. This step is needed to detect semantic dependencies between API requests and meaningfully model them in a vector space.

The second layer, LSTM, is responsible for preserving the cell state and the hidden state of the machine learning model. [23] It operates on a read-write-forget principle. The network learns which information is relevant and will be needed later and which information can be forgotten. The main advantage of the LSTM layer in contrast to classical vanilla RNN is that it solves the vanishing/exploding gradient problem, which appears when passing the gradient recursively for n steps [24].

The third dense layer outputs probabilities over the vocabulary, in our case, the whole API request set, for a

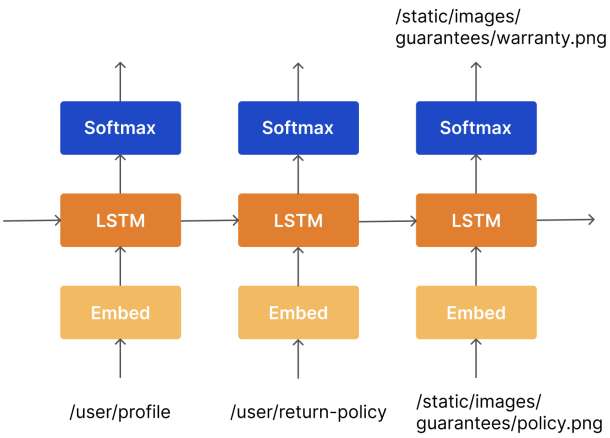

Figure 1: Architecture of RNN

given number of previous API calls. To achieve probabilistic prediction, the softmax activation function was used. For the training configuration, we applied a commonly used categorical cross-entropy as a loss function, adaptive optimizer Adam, and accuracy as a training metric. We used early stop approach to avoid overfitting, this method stops the training process when the accuracy on the validation set starts plateauing. The number of epochs therefore was different, between 5 and 20 epochs, dependent on the window size.

3.1. Data Preprocessing

Due to safety and commercial data considerations, there are limited real data server log datasets available. Many research groups opt to use server logs from their own intranet or record API traffic, creating a server logs dataset specifically for analytical purposes.

In order to test the proposition of being able to predict the next user request, we used an open dataset of server logs of an Iranian online shop, "Online Shopping Store - Web Server Logs" [25]. The dataset comprises more than 10 million logs in Common Log Format (CLF) that Apache uses and contains some valuable information about website usage [26].

Standard CLF server log looks like this:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.gif HTTP/1.0" 200 2326
"http://www.example.com/start.html"
"Mozilla/4.08 [en] (Win98; I ;Nav)"
```

It contains the client's IP address, request time, method used by the client (GET), information on requested resources (/apache_ pb.gif) and the protocol used (HTTP/1.0). The log also contains the respond status code (200), size of the object returned to the client (2326) as well as the referrer and user agent.

For the purposes of demonstrating the idea of predicting the next user request, users are distinguished by a client IP address, and the sessions are limited to a 20-minute time frame. The sliding window technique was applied to create training and test sets. After testing several sizes of the window, the best results were achieved with window size 4, where the fifth request should be predictable from the previous four. The model was also trained using window sizes ranging from 2 to 6, but the larger window sizes led to overfitting and performed worse on a test dataset.

TABLE 1: Dataset Statistics

Number entries	Number unique API requests	Number unique clients	Number identified 20-minute sessions
10 364 865	893 045	258 445	352 296

3.2. Tokenization

Tokenization is the process of converting words into integers for further model training. We used a custom tokenizer in order to be able to treat the whole API request as a single word. The custom tokenizer handles special characters such as underscores and slashes within the request to create a meaningful vocabulary. Sequences are then padded to ensure uniform length of the vectors.

4. Results

In this section, we discuss the initial data and the obtained results.

In the Table 1 you will find statistical metrics of the dataset used.

We tested several window sizes to determine the number of previous requests on which the prediction for the next API request will be based, ranging from 2 to 6. Figure 2 shows the accuracy on training and test sets with regard to window size. The results state that bigger window sizes, such as 5 or 6, clearly lead to overfitting since the accuracy difference on both sets becomes more discrepant. The peak of accuracy on the test set appears when applying the window size, capturing 3 or 4 requests. The exact accuracy values can be found in Table 2.

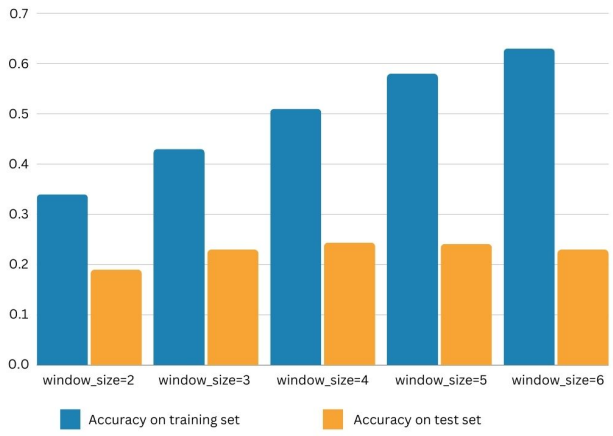

Figure 2: Accuracy on training and test set for different window size parameter

We only focused on the accuracy metric for the API request prediction. The resulting model could not overcome the 24% accuracy on a test set. This might seem low; however, it is worth mentioning that the website,

TABLE 2: Accuracy according to window size

window size	Accuracy on training set	Accuracy on test set
2	34.27%	19.07%
3	43.05%	23.04%
4	53.16%	24.39%
5	58.33%	24.11%
6	62.86%	22.98%

TABLE 3: Prediction Examples

Prediction	Actual
/image/57378/ productModel/100x100	/image/57124/ productModel/100x100
/static/images/ guarantees/warranty.png	/static/images/ guarantees/warranty.png
/static/images/amp/blog.png	/static/images/amp/blog.png

where the API traffic was recorded, is an online shop, containing large amount of dynamic data. The API requests themselves often contain product id, product images paths and other fine-granular details. Here are some examples of such requests:

```
/product/30910
/image/15474?name=1387476275_tc16.jpg&wh=200x200
```

This is, of course, very hard to predict and may be even impossible considering the continuously changing online shop assortment. In the training set, the size of the vocabulary or the token list, in our case the number of distinct API requests, was about 6000 for the training set with 300000 requests. The vocabulary size was dependent on the current batch.

Often, the model would predict the right API endpoint but fail to guess specific resource ID of the product or image. You may find an example of this in the first row of the Table 3.

However, the model shows better performance for the identified patterns, and dynamic data unrelated APIs such as return policy or guarantee resources. (See second and third examples in the Table 3.)

We also tested a hypothesis of building in a threshold on probability predictions, which would only preload data if the probability was sufficient. Unfortunately, this strategy failed to satisfy expectations. Instead, the most often correctly predicted requests were simply the most frequent ones.

Another suggestion on how to utilize the obtained tool could be to only restrict the next API request predictions for certain API endpoints. The most prominent example would be search endpoint. For the user's next search prediction, the larger window size would also be applicable [27].

5. Conclusion and Future Work

In pursuit of our goal to apply a language model approach to API request prediction, we trained a language model on a provided set of logged API calls. The vocabulary of the language model consists of all unique recorded API requests, and the word sequences are modeled based on user sessions.

Due to the large number of possible API URLs and dynamic data such as product IDs or image source ID, our effort did not yield impressive results. However, it still provided valuable insights into the future possibilities and limitations of the API request prediction. The model was able to detect some reoccurring patterns and learn the semantics of the API endpoint. This approach could be used for websites with a static structure and a limited number of API endpoints.

Some possible solutions to the low accuracy problem could be limiting the model to a certain set of API dynamic data-insensitive endpoints. The main drawback remains the fact that each website would have to train its own machine learning model based on the API structure.

In the future, the idea of successively guessing the next resource endpoint of the API URL should be examined. In this approach, the API paths would not be treated as single tokens but could be split up into hierarchical resource endpoints. It could exploit the hierarchical URL path structure.

References

[1] D. S. Berger, "Towards lightweight and robust machine learning for cdn caching," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, ser. HotNets '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 134–140. [Online]. Available: https://doi.org/10.1145/3286062.3286082

[2] E. Ghabashneh and S. Rao, "Exploring the interplay between cdn caching and video streaming performance," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 516–525.

[3] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "Effective Prediction of Web-user Accesses: a Data Mining Approach," Aug. 2001.

[4] D. S. Berger, "Towards Lightweight and Robust Machine Learning for CDN Caching," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. Redmond WA USA: ACM, Nov. 2018, pp. 134–140. [Online]. Available: https://dl.acm.org/doi/10.1145/3286062.3286082

[5] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, 2002.

[6] R. Cooley, B. Mobasher, and J. Srivastava, "Web mining: information and pattern discovery on the World Wide Web," in *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*. Newport Beach, CA, USA: IEEE Comput. Soc, 1997, pp. 558–567. [Online]. Available: http://ieeexplore.ieee.org/document/632303/

[7] M. A. T. Pratama and A. T. Cahyadi, "Effect of user interface and user experience on application sales," *IOP Conference Series: Materials Science and Engineering*, vol. 879, no. 1, p. 012133, jul 2020. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/879/1/012133

[8] J. B. Schafer, J. Konstan, and J. Riedl, "Recommender systems in e-commerce," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 158–166.

[9] M. Géry and H. Haddad, "Evaluation of web usage mining approaches for user's next request prediction," in *Proceedings of the 5th ACM international workshop on Web information and data management*. New Orleans Louisiana USA: ACM, Nov. 2003, pp. 74–81. [Online]. Available: https://dl.acm.org/doi/10.1145/956699.956716

[10] W. Gaul and L. Schmidt-Thieme, "Mining Web Navigation Path Fragments," Aug. 2000.

[11] B. Nigam, S. Tokekar, and S. Jain, "Evaluation of Models for Predicting User's Next Request in Web Usage Mining," *International Journal on Cybernetics & Informatics*, vol. 4, no. 1, pp. 01–12, Feb. 2015. [Online]. Available: http://www.airccse.org/journal/ijci/papers/4115ijci01.pdf

[12] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber, "Pushing cdn-isp collaboration to the limit," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, p. 34–44, jul 2013. [Online]. Available: https://doi.org/10.1145/2500098.2500103

[13] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, Nov. 2016, pp. 50–56. [Online]. Available: https://doi.org/10.1145/3005745.3005750

[14] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "{AdaptSize}: Orchestrating the Hot Object Memory Cache in a Content Delivery Network," 2017, pp. 483–498. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/berger

[15] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," Jan. 2019, arXiv:1709.06560 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1709.06560

[16] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," Oct. 2017, arXiv:1710.02298 [cs]. [Online]. Available: http://arxiv.org/abs/1710.02298

[17] M. Lecuyer, J. Lockerman, L. Nelson, S. Sen, A. Sharma, and A. Slivkins, "Harvesting Randomness to Optimize Distributed Systems," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. Palo Alto CA USA: ACM, Nov. 2017, pp. 178–184. [Online]. Available: https://dl.acm.org/doi/10.1145/3152434.3152435

[18] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, "Learning distributed caching strategies in small cell networks," in *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*. Barcelona, Spain: IEEE, Aug. 2014, pp. 917–921. [Online]. Available: http://ieeexplore.ieee.org/document/6933484/

[19] "Faulty reward functions in the wild." [Online]. Available: https://openai.com/research/faulty-reward-functions

[20] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *arXiv preprint arXiv:1708.04133*, 2017.

[21] S. R. A and B. Rudra, "Evaluation of recurrent neural networks for detecting injections in api requests," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 2021, pp. 0936–0941.

[22] M. Arivukarasi and A. Antonidoss, "Performance analysis of malicious url detection by using rnn and lstm," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, 2020, pp. 454–458.

[23] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[24] ——, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[25] F. Zaker, "Online Shopping Store - Web Server Logs," May 2021. [Online]. Available: https://doi.org/10.7910/DVN/3QBYB5

[26] "Log Files - Apache HTTP Server Version 2.4." [Online]. Available: https://httpd.apache.org/docs/2.4/logs.html

[27] A. Agarwal, "Predicting the next search keyword using Deep Learning | by Atul Agar…," May 2023. [Online]. Available: https://archive.ph/vGwEV