# CS559 Final Report

**Team member - Mitul Vardhan(12141070)**

**Github Link: https://github.com/ramanujan123/CS559_PROJECT**

## Project Title : Implementation and Evaluation of ML-based Caching Algorithm

## Introduction

Cache replacement algorithms have undergone significant evolution over time, with each iteration aiming to mitigate certain deficiencies found in its predecessors. Nonetheless, despite considerable progress, contemporary caching algorithms still present opportunities for enhancement. Recent advances in machine learning open up new and attractive approaches for solving classic problems in computing systems. For storage systems, cache replacement is one such problem because of its enormous impact on performance.

There are some learning based algorithms for caching like LeCaR, which has access to two simple policies,LRU and LFU.  LeCaR uses regret minimization, a machine learning technique that allows the dynamic selection of one of these policies upon a cache miss. But, it has been shown that LeCaR underperforms many state-of-the-art algorithms such as ARC and  LIRS for many production workloads.

In this project, an attempt has been made to design an adaptive learning cache replacement algorithm, which will perform better than the state-of-the-art. Simulation experiments have been conducted on industry-level workloads and findings have been comprehensively showcased. Also,  a mirror of wikipedia.org was made by running a caching Nginx Proxy in front of it. This would help reduce latency and availability of wiki contents even in the times of DDoS attacks.

## Design of Learning Based Caching Algorithm

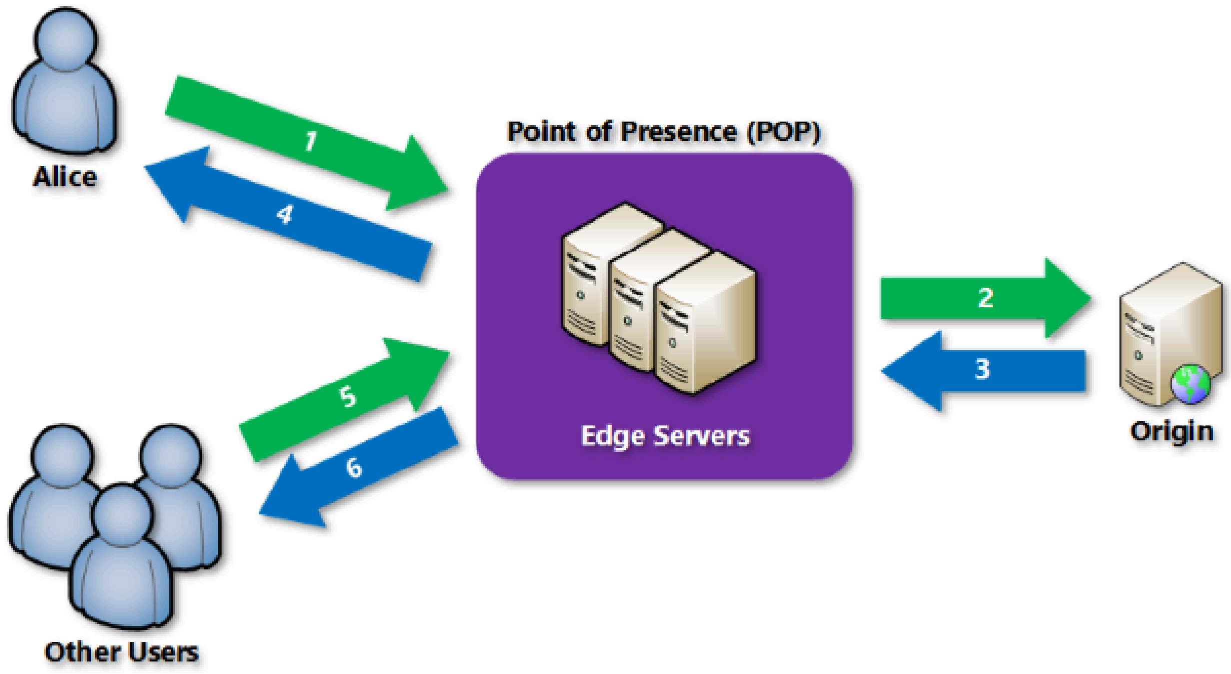
- **Initialization:**
    - Initialize cache with a size of N.
    - Initialize learning rate randomly between 10^-3 and 1.
    - Define the window size for monitoring performance.
- **Gradient-based Stochastic Hill Climbing with Random Restart:**

    - At the end of every window of N requests, calculate the gradient of performance (average hit-rate) with respect to the learning rate over the previous two windows.
    - Use a gradient-based stochastic hill climbing approach with random restart to adjust the learning rate.
- **Direction of Learning Rate Adjustment:**
    - If the gradient is positive, sustain the direction of change of the learning rate. If negative, reverse the direction.
    - The direction of change determines whether to increase or decrease the learning rate.
- **Magnitude of Learning Rate Adjustment:**
    - The amount of change in learning rate from the previous window determines the magnitude of the change in learning rate for the next window.
    - If the performance increases by increasing the learning rate, increase the learning rate by multiplying it by the amount of learning rate change from the previous window, and vice versa.
- **Monitoring Performance:**
    - If the learning rate doesn't change for consecutive windows and the performance degrades continuously or becomes zero, record this.
    - If the performance degrades for 10 consecutive window sizes, reset the learning rate to the initial value.
    - The objective is to restart learning when performance drops for a longer period.
- **Algorithm Loop:**
    - Repeat steps 2-5 for each window of N requests until the termination condition is met.

Please refer to the GitHub repo for the full implementation of the algorithm.

## Computer System Design

In this project, a learning based caching algorithm has been designed and is used to cache wikipedia.org using a virtual private server (VPS).
High Level Design of the system is given below:



Wikipedia's infrastructure provides regular database dumps and static HTML archives to the public, and has permissive licensing that allows for rehosting with modification. In this project, these publicly available dumps have been used to host Wikipedia mirrors.

A Nginx caching proxy was created in front of wikipedia.org and it would serve the clients. If the user's request is present in the cache server, it will be served via proxy only. Otherwise, the request will be sent to the original wikipedia server. In this way, it will help in reducing the latency and enhancing user experience. It also has the benefit that if some DDoS attacks happen on the original wikipedia, then the user's request can be served from the proxy server if it is present in the cache.

**Note:** Due to unavailability of Cloud Platforms, proxy server in this project was established on local machine's virtual machine(VM).

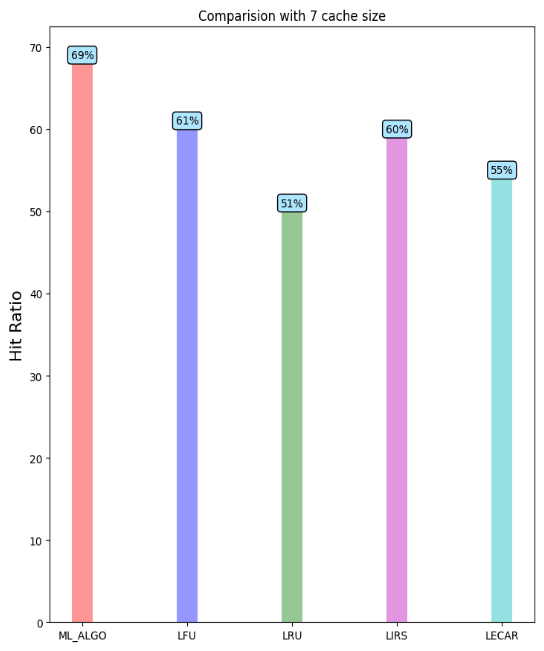
## Simulation Experiment and Comparison with SOTA Algorithms:

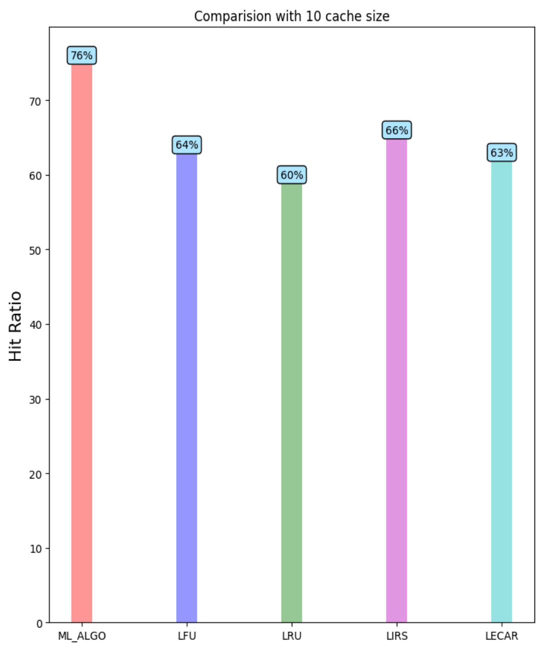
The algorithm was simulated on the following workloads:

1. MSR Cambridge :
2. FIU SRC_Map
3. CloudVPS

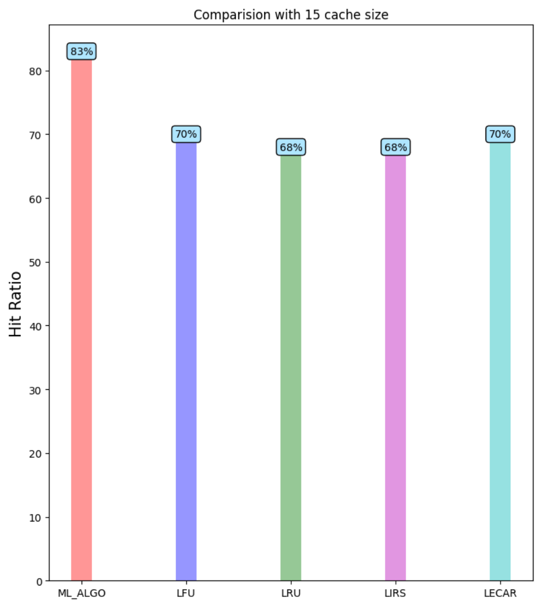
The results are  shown below:

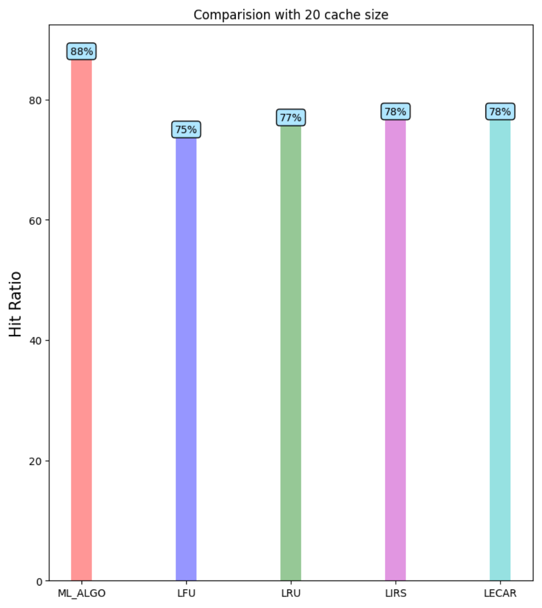
*when learning rate was initialized at 0.05 and cache size was set at 7 and 10 :*



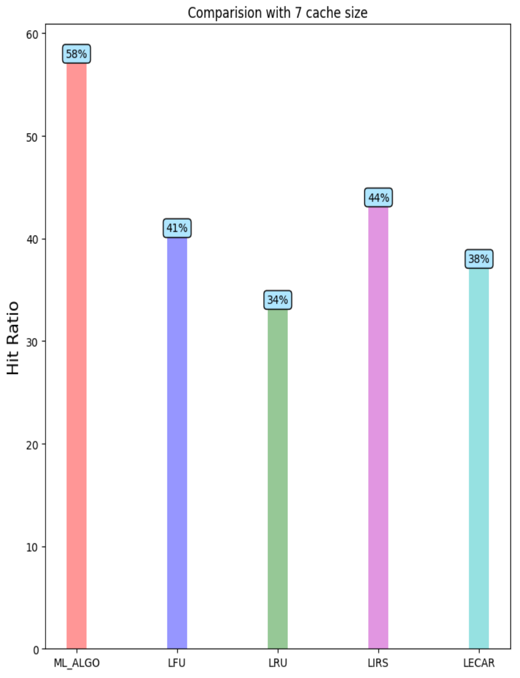


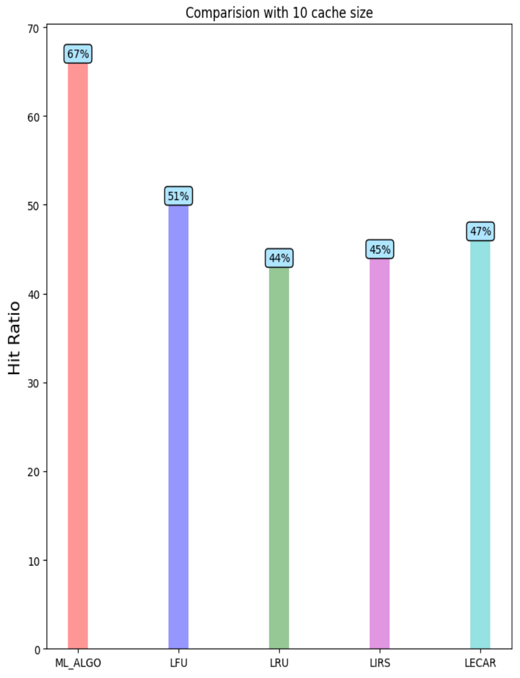
*when learning rate was initialized at 0.05 and cache size was set at 15 and 20 :*



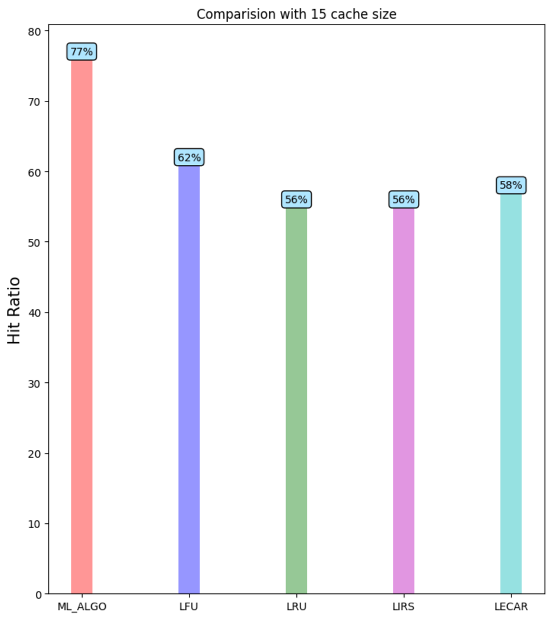


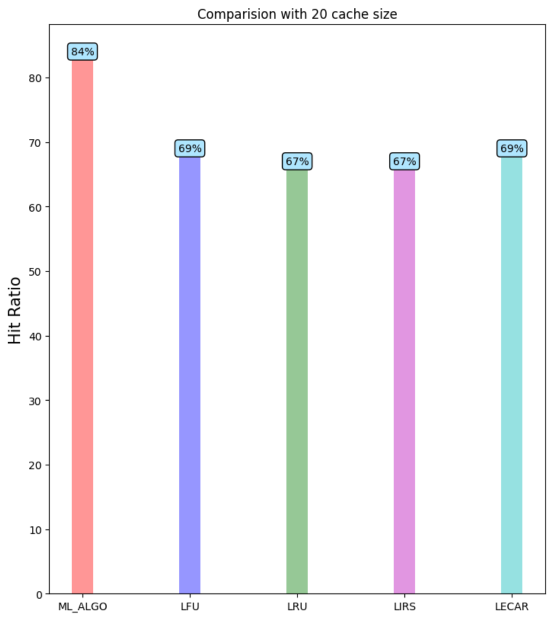
*when the learning rate was initialized at 0.1 and cache size was set at 7 and 10:*



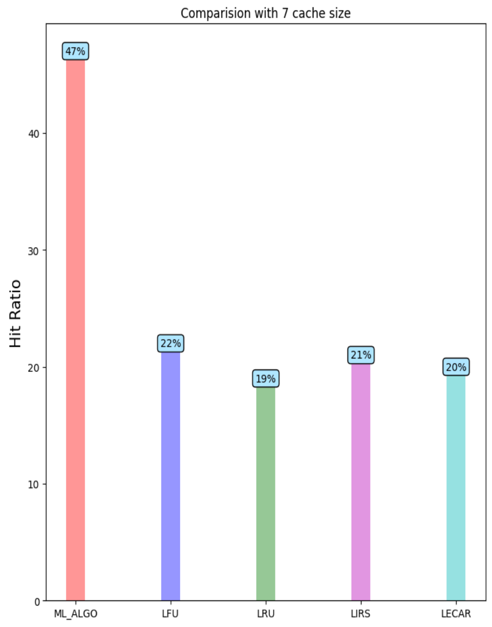


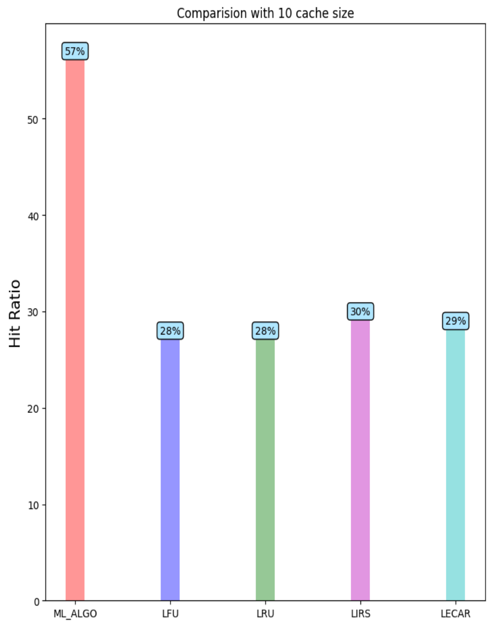
*when the learning rate was initialized at 0.1 and cache size was set at 15 and 20:*



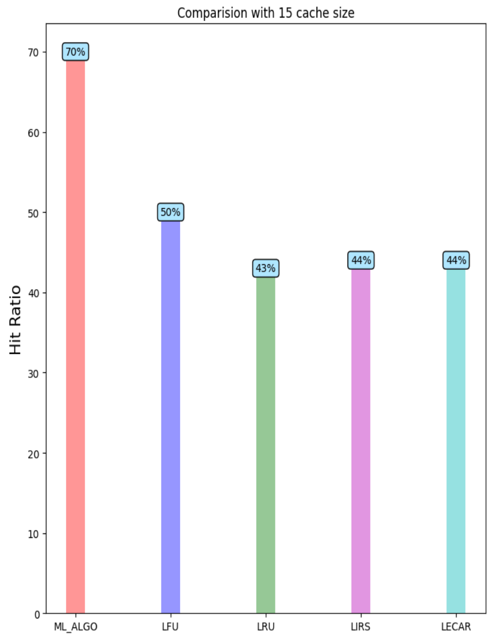


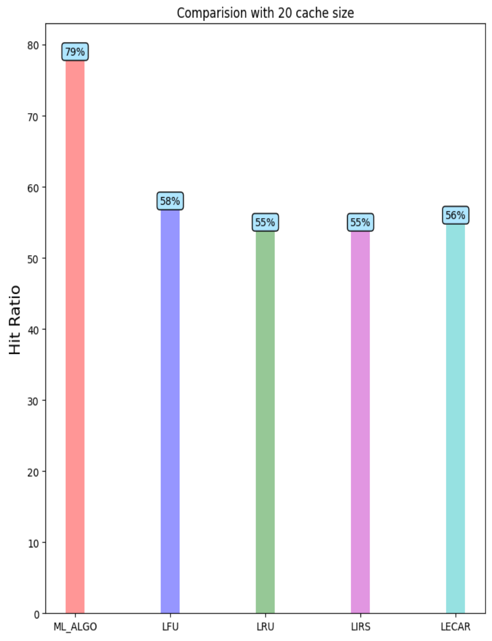
*when the learning rate was initialized at 0.3 and cache size was set at 7 and 10:*





*when the learning rate was initialized at 0.3 and cache size was set at 15 and 20:*





This is the result obtained for **MSR Cambridge** workload. Learning based algorithm was compared with many state of the art algorithms and at varying learning rates. It can be seen that the Learning based caching algorithm has performed better than many standard algorithms.

Note: All these results are also available in the github repo.

## <u>Operational Issues Faced</u>

1. **Reward Design and Function:**
   Designing an effective reward function was challenging as it  may lead to suboptimal behavior if not carefully crafted.
   **Solution:** Experimented with different reward shaping techniques to provide more informative and stable feedback to the RL agent during training.

2. **Training Stability and Convergence:**
   RL algorithms are sensitive to hyperparameters, training setup, and environmental dynamics, leading to instability or slow convergence during training.
   **Solution:** Regularized the learning model with techniques like  gradient clipping to prevent divergence or instability.

3. **Dependency on Local Virtual Machines due to Lack of Cloud Computing Platform Access:**
   Inability to access cloud platforms, which  require credit card credentials for provisioning Virtual Private Servers (VPS), necessitated the use of local virtual machines (VMs) to host the server environment. This introduced additional complexities related to resource management, networking configuration and security.
   **Solution:** Optimize resource allocation and management for both the host server and guest to ensure smooth operation without resource contention.

## REFERENCES

[ 1 ]  Y. Im, P. Prahladan, T. H. Kim, Y. G. Hong and S. Ha, "SNN-cache: A practical machine learning-based caching system utilizing the inter-relationships of requests," 2018 52nd Annual Conference on Information Sciences and Systems (CISS), Princeton, NJ, USA, 2018, pp. 1-6, doi: 10.1109/CISS.2018.8362281.
keywords: {Metadata;Servers;Heuristic algorithms;Correlation;Machine learning algorithms;Data mining;Real-time systems},

[ 2 ]  N. Zaidenberg, L. Gavish and Y. Meir, "New caching algorithms performance evaluation," 2015 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Chicago, IL, USA, 2015, pp. 1-7, doi: 10.1109/SPECTS.2015.7285291.
keywords: {Benchmark testing;Algorithm design and analysis;Classification algorithms;Indexes;Arrays;Servers;Complexity theory;Caching algorithms;memcached;ARC;RMARK},

[ 3 ]  Rodriguez, L. V., Yusuf, F. B., Lyons, S., Paz, E., Rangaswami, R., Liu, J., Zhao, M., and Narasimhan, G. Learning cache replacement with cacheus. In FAST (2021), pp. 341–354.

[ 4 ]  Jiayi Chen, Nihal Sharma, Tarannum Khan, Shu Liu, Brian Chang, Aditya Akella, Sanjay Shakkottai, and Ramesh K Sitaraman. 2023. Darwin: Flexible Learning-based CDN Caching. In Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23). Association for Computing Machinery, New York, NY, USA, 981–999. https://doi.org/10.1145/3603269.3604863

[5]  https://docs.sweeting.me/s/self-host-a-wikipedia-mirror and https://github.com/pirate/wikipedia-mirror (used for creating Nginx server to host wikipedia.org)

[6] https://learn.microsoft.com/en-us/azure/cdn/media/cdn-overview/cdn-overview.png