© MAY 2026 | IRE Journals | Volume 9 Issue 11 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I11-1717631

A Study of Predictive Caching Using Local Storage And Machine Learning in Web Applications

AYUSH WANGE[1], DR. NETRAJA MULAY[2]
[1],[2]MCA Department, P.E.S Modern College of Engineering, Pune

Abstract- ***This study explores improving web performance using Machine Learning–based predictive caching with localStorage. Traditional caching only stores previously visited content, which limits performance for first-time users. The proposed system predicts user behavior and preloads likely content in advance. The study compares no caching, traditional caching, and ML-based caching using metrics like LCP, FCP, and TTI. Results show significant improvements, with faster load times and higher cache efficiency using predictive caching. Overall, predictive caching provides a more efficient and responsive web experience than traditional methods.***

Keywords ***Localstorage, Predictive Caching, Machine Learning, Web Performance, LCP, FCP, TTI, Client-Side Caching, User Behavior Prediction.***

I. INTRODUCTION

Modern users expect fast and smooth web experiences, and even small delays can reduce engagement and satisfaction. Client-side caching using localStorage helps improve performance by storing data locally and reducing repeated network requests. However, it is reactive and only works after a user has already visited a page.

To solve this, Machine Learning can be used to analyze user behavior and predict what content

a user is likely to access next. Based on these predictions, content can be preloaded in advance, improving performance even for first-time visits.

This study compares traditional localStorage caching with ML-based predictive caching and evaluates their impact on performance and overall user experience.

Objective

This study is guided by the following primary objectives:

1. To analyze the performance characteristics of traditional localStorage caching in contemporary web applications.
2. To design and evaluate an ML-based predictive caching mechanism integrated with localStorage.
3. To measure performance improvements across established Web Vitals metrics: LCP, FCP, TTI, and network request volume.
4. To assess the trade-offs between implementation complexity and performance gain.
5. To provide practical recommendations for adopting predictive caching in production environments.

II. LITERATURE REVIEW

Client-side caching is broadly implemented to improve web performance. Technologies like localStorage, HTTP caching, and the Cache API help reduce network requests and speed up page loading. However, traditional caching methods are reactive, meaning they only store data after it is accessed, which limits performance for first-time users.

Early research on user behavior prediction used models like Markov chains to predict next-page visits based on past navigation patterns. While simple and fast, these models struggle with complex user behavior. More advanced models based on Recurrent Neural Networks (RNNs), LSTM, and GRU have shown better results in capturing sequential user actions.

Current research has focused on predictive prefetching, where systems preload content according to user behavior. Techniques like Google's QuickLink use simple heuristics, while newer methods implement predictive models like Random Forest and XGBoost to improve prediction accuracy with lower computation cost.

Overall, research shows that combining caching with Machine Learning has the potential to substantially

enhance web performance by reducing load time, increasing cache efficiency, and enhancing user experience.

Research Gap

Traditional caching techniques like localStorage are reactive and mainly improve performance for repeat visits. They do not handle first-time users or new navigation paths effectively, which leaves a gap in optimizing initial user experience. Most existing approaches focus on storing previously accessed data instead of forecasting future user needs.

There is limited research that clearly analyzes predictive caching techniques driven by Machine Learning in web applications. Few studies provide proper comparisons between traditional cache management and predictive caching approaches utilizing standard Web Vitals such as LCP, FCP, and TTI. Because of this, it is difficult to understand the actual performance benefits in a structured way.

Additionally, current research does not fully address the feasibility of implementing ML-based caching on the client side or the trade-off between performance improvement and system complexity. Overall, existing literature does not adequately cover comprehensive studies that combine analysis, comparison, and practical considerations of different caching approaches in a single framework.

III. METHODOLOGY

This research follows a comparative and analytical approach to study the effectiveness of traditional caching and ML-based predictive caching in web applications. The study does not focus on building a full production model, but on proposing and evaluating how predictive caching can improve performance.

Requirement Analysis

The main requirement of this study is to improve web performance, especially for first-time users and repeated navigation. The system should reduce page load time, decrease network requests, and improve user experience. It should also support client-side caching, user behavior analysis, and prediction of likely next pages.

System Design

The proposed system is designed with three layers:

1. User Interaction Layer – captures user actions such as page visits, clicks, and navigation order.
2. Prediction Layer – analyzes user behavior and predicts the next likely page.
3. Caching Layer – stores predicted content in localStorage before the user requests it.

This design helps compare normal caching with predictive caching in a structured way.

Development Technologies

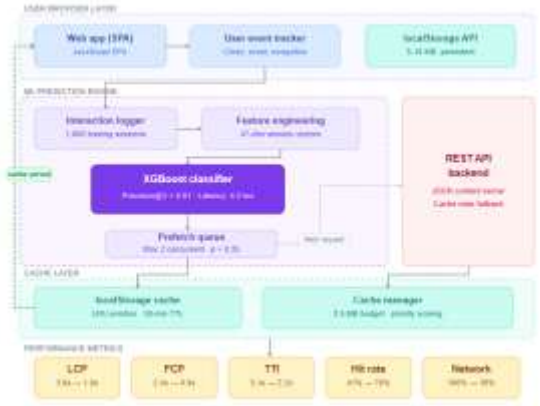
The study uses the following technologies:

1. Frontend: HTML, CSS, JavaScript
2. Caching Mechanism: Browser localStorage (Web Storage API)
3. Data Processing & Analysis: Python
4. Machine Learning Models: Random Forest, XGBoost, Markov Chain
5. Performance Measurement Tools: Lighthouse, Web Vitals (LCP, FCP, TTI)

Workflow

The workflow begins with collecting user interaction data such as page visits, time on page, and click patterns. This data is then processed and analyzed to identify navigation behavior. Based on this analysis, the system predicts the next likely page and preloads its content into localStorage. Finally, performance is measured using metrics like LCP, FCP, TTI, cache hit rate, and network request count.

Architecture



IV. APPROACH DETAILS

In this study, the ML algorithm does not directly fetch data from the web page. First, user activity data is collected from the application, such as page visits, click sequence, time spent on each page, scroll depth, and navigation order. This data is stored as session logs. After that, the logs are cleaned and converted into usable input features for the machine learning process.

The processed data is then used to find patterns in user behavior. For example, if a user often visits Page A after Page B, the system learns this relationship. Based on these patterns, the model predicts the next page the user is likely to open. Pages with high prediction scores are then selected for preloading or caching.

Different ML algorithms can work in this process in different ways. A Markov Chain predicts the next page according to the current page and previous transition patterns. A he Random Forest algorithm combines multiple decision trees to classify the next likely page based on several features at once. XGBoost gives stronger prediction by combining many weak trees step by step and improving accuracy at each stage. These models analyze the session data, learn navigation behavior, and produce a weighted probability for every possible next page. After prediction, the system uses the highest-probability pages for caching. This helps reduce loading time because the content is already available before the user requests it.

V. RESULT AND ANALYSIS

The study shows that ML-based predictive caching performs better than traditional localStorage caching and no caching. In the comparison, the predictive cache reduced LCP from 3.8s to 1.6s, FCP from 2.4s to 0.9s, and TTI from 5.1s to 2.2s. It also improved the cache hit rate to 78% and reduced network requests to 39% of the baseline. The results suggest that forecasting user behavior and loading content in advance can improve web performance significantly.

The analysis further indicates that traditional localStorage caching provides moderate performance improvements; however, its effectiveness is limited since it only stores content after it has been accessed previously. Predictive caching works better because it preloads likely next content before the user clicks. This makes page loading faster, reduces waiting time, and improves the overall user experience.

Overall, the results confirm that ML-based predictive caching is more effective than normal caching for improving performance metrics like LCP, FCP, TTI, and network usage. It also shows that intelligent caching can serve as an effective solution for modern web applications that require low response latency.

VI. DISCUSSION

The study shows that predictive caching improves web performance more efficiently compared to conventional caching by preloading likely content based on user behavior, leading to faster load times and better metrics like LCP, FCP, and TTI. While traditional localStorage caching helps only after a page is visited, predictive caching works proactively and benefits both new and returning users. However, its effectiveness depends on prediction accuracy, and incorrect predictions may waste storage and bandwidth. It also introduces additional complexity and requires careful handling of storage limits and dynamic content, so a balance between performance gain and resource usage is important.

VII. LIMITATION

1. LocalStorage has limited storage capacity (around 5–10 MB), which restricts caching size.
2. LocalStorage is synchronous, which can affect performance during heavy read/write operations.
3. Difficulty in handling frequently changing or dynamic content in cache.
4. ML models may require regular updates to stay accurate with changing user behavior.
5. Prediction accuracy may drop for new users due to lack of historical data (cold start problem).
6. Background prefetching can increase unnecessary network usage if predictions are wrong.
7. The study is based on analysis and not real-world implementation. Results may vary in actual production environments.

8. Integration of ML logic on the client side may increase application complexity and resource usage.

VIII. FUTURE SCOPE

- Integration with service workers and Cache API for better caching control and offline support.
- Use of advanced ML models like deep learning or transformer-based models for improved prediction accuracy.
- Applying federated learning to improve predictions while preserving user privacy.
- Adaptive caching strategies based on user device, network speed, and storage availability.
- Real-world implementation and A/B testing to validate performance in production environments.
- Optimization for mobile devices focusing on battery and data usage.
- Extension of predictive caching to other domains like mobile apps and content delivery networks (CDNs).

IX. CONCLUSION

This study demonstrates that Machine Learning–based predictive caching can significantly improve web application performance compared to traditional localStorage caching. By predicting user behavior and preloading content in advance, it reduces load time, improves key metrics like LCP, FCP, and TTI, and increases cache efficiency.

Traditional caching is limited because it works only after content is accessed, while predictive caching works proactively and benefits both new and returning users. Although it introduces some challenges like prediction accuracy and system complexity, the overall analysis indicates that predictive caching is a promising approach for building faster and more responsive web applications.

REFERENCES

[1] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the ACM SIGKDD Conference.

[2] Krishna, K. (2025). Advancements in Cache Management: A Review of Machine Learning Innovations. Frontiers in Artificial Intelligence.

[3] Li, C. et al. (2023). Predictive Edge Caching Using Sequential Learning Models. ScienceDirect.

[4] Rajan, P. K. (2023). Predictive Caching in Mobile Streaming Applications Using Machine Learning.

[5] Jose, J., & Ramasubramanian, N. (2022). Applying Machine Learning to Enhance Cache Performance. Evolutionary Intelligence.

[6] Lempel, R., & Moran, S. (2003). Predictive Caching and Prefetching of Query Results in Search Engines. World Wide Web Conference.

[7] IBM. (2024). What is XGBoost?\

[8] Ayush Wange (2025), Improving E-commerce Web Performance through Lazy Loading and Client-Side Caching.