SPECIAL ISSUE PAPER

On the Use of Embedding Techniques for Modeling User Navigational Behavior in Intelligent Prefetching Strategies

Tolga Buyuktanir | Mehmet S. Aktas

Computer Engineering Department, Yildiz Technical University, Istanbul, Turkiye

Correspondence: Tolga Buyuktanir (tolga.buyuktanir@std.yildiz.edu.tr)

Received: 15 August 2024 | **Revised:** 24 November 2024 | **Accepted:** 28 November 2024

Keywords: browsing behavior modeling | client-server systems | data delivery | data-intensive applications | graph embedding | predictive prefetching | user navigational behavior

ABSTRACT

In today's data-intensive client-server systems, traditional caching methods often fail to meet the demands of modern applications, especially in mobile environments with unstable network conditions. This research addresses the challenge of improving data delivery by proposing an advanced prefetching framework that utilizes various embedding techniques. We explore how to model user navigation using graph-based, autoencoder-based, and sequence-to-sequence-based embedding methods and assess their impact on prefetching accuracy and efficiency. Our study shows that utilizing these embedding techniques with supervised learning models improves prefetching performance. We also present a software architecture that blends supervised and unsupervised learning approaches, along with user-specific and collective learning models, to create a robust prefetching mechanism. The contributions of this study include developing a scalable prefetching solution using machine learning/deep learning algorithms and providing an open-source prototype of the proposed architecture. This paper offers a significant improvement over previous research and provides valuable insights for enhancing the performance of data-intensive applications.

1 | Introduction

The rapid increase of data-intensive client-server systems has made the rapid delivery of data to end-users a major challenge [1, 2]. Traditional storage systems, although effective to some extent, often fail to meet the dynamic requirements of modern applications, especially for mobile environments. This requires developing intelligent ways to capture data before users ask for it, thus making users more efficient with the system [3].

An important concern in data-intensive client-server systems is the substantial data traffic and the duration needed to handle this data. Within these systems, each user request initiates the processing of extensive data sets on the server. This operation requires substantial resources and results in an increased workload on the server, resulting in delays and a decline in performance [4]. Prefetching algorithms can mitigate this load by strategically loading frequently needed data, thus reducing processing latencies and enhancing overall system performance [5].

In mobile applications, user experience is highly dependent on the quality of network connectivity. Mobile devices often operate under variable and frequently poor network conditions, which slow down data access and loading times, adversely affecting the user experience. Prefetching can provide a solution by downloading the necessary data in advance, ensuring a fast and seamless experience regardless of network quality. This capability can significantly enhance user satisfaction and the usability of mobile applications [4, 6–8].

Customers are often impatient and have ever-increasing expectations. They demand instant responses and uninterrupted performance. Traditional caching systems may fall short of meeting these expectations as they rely on historical data and do not predict future requests. Prefetching, on the other hand, enhances application performance and responsiveness by anticipating and preloading data based on predicted user actions, leading to a more satisfying user experience [9].

Data security and privacy can also contribute to performance issues. In data-intensive applications, various encryption and authentication processes are used to ensure data security and privacy. These processes can slow down data access and transmission. Prefetching can speed up these security processes by completing them in advance, allowing users to access data quickly and securely. This not only improves performance but also reduces user concerns regarding security [10].

In the literature, there is a noticeable lack of studies that leverage advanced modeling techniques such as autoencoders, LSTM, and graph-based encoding approaches to model users' navigational behavior for supporting prefetching mechanisms. While traditional methods have made some progress in predicting user actions, they often rely on simpler embedding techniques like word2vec [7]. These advanced approaches, however, have the potential to capture more complex patterns in user behavior, leading to more accurate and efficient prefetching strategies. The absence of such sophisticated modeling in existing research highlights a significant gap that this study aims to address [11–16].

Also, there aren't enough system resources for prefetching that use supervised learning AI models [17]. These models learn from data about each user's browsing behavior and all users' behavior combined. Most current methods only look at data about a single user or make assumptions based on data from a larger set of users, without mixing the two in a useful way [18–20]. This dual-learning method can make prefetching predictions a lot more accurate by adapting them to the specific trends of each user while also taking into account how all users behave as a whole. Another important gap is that there aren't many complete studies that use these kinds of mixed supervised learning models for prefetching [7, 8].

The literature has not extensively explored both supervised and unsupervised learning methods in prefetching systems. While there are some cases where either supervised or unsupervised methods are used, integrating both approaches to create a more resilient and flexible prefetching mechanism is rarely discussed [8]. Hybrid systems that use unsupervised learning to understand overall patterns and supervised learning to make accurate predictions can significantly improve performance. This study aims to fill this gap by presenting a framework that combines these approaches for intelligent prefetching [7].

This research addresses these gaps by proposing a comprehensive framework that leverages advanced modeling techniques, including autoencoders, LSTM, and graph-based encoding approaches, to model users' navigational behavior. By incorporating these sophisticated methods, our study captures complex patterns in user behavior, leading to more accurate and efficient prefetching strategies.

To assess the performance of our proposed prefetching approach, we conducted baseline comparisons with our previous work, where unsupervised learning techniques, specifically PrefixSpan, were applied to enhance prefetching performance [8]. In that study, we reported significant improvements in prefetching accuracy and cache hit rates compared to a no-prefetching scenario. In this study, we have utilized both the no-prefetching condition and the unsupervised learning method from our prior work as baseline references to evaluate the current dataset. The results are promising: our proposed supervised learning approach outperforms the unsupervised approach in critical metrics, such as cache hit rates. This comparison highlights that while unsupervised learning provides substantial improvements over no-prefetching strategies, the supervised learning techniques introduced in this study yield even greater enhancements.

Furthermore, we introduce a dual learning approach that utilizes supervised learning AI models trained on both user-specific navigational behavior data and aggregated data from all users. This approach enhances the accuracy of prefetching predictions by tailoring them to individual user patterns while benefiting from the collective behavior of all users. Additionally, we integrate both supervised and unsupervised learning methods in a hybrid manner, creating a robust and adaptive prefetching mechanism that leverages the strengths of both approaches. This hybrid system improves the understanding of general patterns through unsupervised learning and makes specific predictions through supervised learning. In conclusion, our study not only fills the current knowledge gaps in the literature but also establishes an entirely novel framework for future research. We have developed a scalable and adaptable software architecture specifically designed for data-intensive client-server systems. This architecture greatly reduces latency and improves user experience, especially in mobile applications.

The rest of the paper is organized as follows: Section 1 is an introduction and provides an overview of the research context, highlighting the significance of predictive prefetching in client-server systems and the motivation behind this study. Section 2 reviews existing literature on predictive prefetching models, particularly focusing on navigational behavior modeling approaches and the use of embedding techniques. In Section 3, we outline the specific challenges addressed by this research. Section 4 details the approach taken in this study, describing the techniques used to model user behavior, the embedding methods employed, and the machine learning models developed for next-page-visit prediction. Section 5 provides a thorough explanation of the system architecture, data processing steps, and the tools used for model training and evaluation. In Section 6, we present the outcomes of the experiments conducted, analyzing the performance of the proposed models in terms of prediction accuracy and cache miss reduction. Finally, Section 7 concludes the key findings of the study, discusses the implications of the results, and suggests potential directions for future research in this area.

2 | Related Work

Prefetching and caching are powerful techniques to enhance the performance and user experience of data-intensive web and mobile applications. Prefetching is the process of loading data into a cache by predicting the data that the user will request in the future. Caching stores frequently accessed data in memory to reduce access times and minimize latency [4, 8, 21, 22].

These techniques are vital for mitigating data access latency and bandwidth limitations. By prefetching and caching data, systems can significantly reduce response time, resulting in a regular and more responsive user experience. Moreover, these methods decrease server load, as repeated data requests can be served from the cache, optimizing resource utilization and minimizing operational costs [4, 8, 21].

Recent research in caching and prefetching for web and mobile applications has focused on adaptive and intelligent strategies leveraging machine learning and user behavior analysis. These approaches aim to improve prefetching accuracy and caching efficiency [8, 23].

Machine learning (ML) is playing a big role in improving prefetching and caching by making predictions more accurate through the analysis of complex user behavior. Studies show that ML-based prefetching can better handle small changes in how users interact with a system, allowing it to store useful data in advance more effectively than traditional methods that rely on simple rules. Techniques like reinforcement learning, supervised learning, and deep learning are used to study user behavior in real time, helping prefetching models quickly adapt to both short-term and long-term patterns. Adding ML to caching has also been shown to reduce cache misses and make better use of resources, making ML-driven caching an important area of research in prefetching technology [5, 7, 8, 14].

Prefetching and caching are useful for making applications respond faster, but they have limits that can make them hard to scale and use efficiently. One main issue is predicting what users will need next, especially in environments where users' actions change frequently. The system may fetch unnecessary data if predictions are wrong, using up storage space and network resources. Caching also has storage limits, which can cause frequent clearing of the cache and lower success rates, especially for apps with a lot of different, constantly changing content. Studies show that the high resource use and extra costs from wrong predictions and clearing outdated data can reduce the benefits of these methods, especially in large systems [4, 24, 25].

Modeling user behavior is fundamental to effective prefetching and caching strategies. User behavior can be represented as data encapsulating various aspects of user interactions, such as navigation patterns, content preferences, and temporal usages. User behavior data can be represented as vectors. There are methods in the literature that successfully represent user behavior [11–13, 26–29]. Several techniques exist for creating user behavior vectors. Traditional methods include frequency and recency-based models, capturing how often and recently a user accessed certain data [8]. Our proposed approaches utilize embeddings, transforming user behavior into continuous vector spaces. These embeddings capture complex relationships and patterns in user behavior, enabling more accurate predictions.

Word embeddings, which is a natural language processing method, such as Word2Vec [30] and GloVe [31] encode words as vectors within a continuous vector space. These embeddings capture semantic relationships between words by analyzing their co-occurrence in extensive text corpora. In the context of user behavior modeling, analogous techniques are employed to represent user actions or content items as vectors, thereby encapsulating their usage patterns and interrelationships [12, 32, 33]. Word embeddings are superior at encoding the semantic relationships between words based on their co-occurrence in text. For this reason, in our study, we created embedding vectors of user navigational data with the word2vec method.

Node2Vec [34] and DeepWalk [35] are prominent graph-based embedding methods that learn low-dimensional representations of graph nodes through random walks. Node2Vec excels in capturing the structural relationships between nodes by optimizing a neighborhood-based objective function, which is particularly useful for modeling user interactions and content similarities. DeepWalk, on the other hand, uses a skip-gram model to capture both local and global graph structures, learning embeddings from random walks generated over the graph [36].

In addition to these methods, Graph2Vec [37] techniques, including Struct2Vec, focus on converting entire graph structures into vector representations. Struct2Vec [38] captures structural information by analyzing node attributes and relationships, offering a more nuanced understanding of graph-based data. These graph-to-vector (Graph2Vec) approaches facilitate machine learning applications such as node classification, link prediction, and clustering. By improving the modeling of complex networks and user behaviors, these techniques lead to more accurate predictions and personalized recommendations. In our study, we used Node2vec and DeepWalk methods from Graph2Vec methods to extract user navigational data's structural and relational information embeddings.

The Sequence-to-Sequence [39, 40] model, an encoder-decoder architecture, is commonly employed in tasks such as machine translation. This architecture is also applicable for creating user behavior embeddings. In this context, the encoder processes the input sequence and generates the corresponding embeddings. User navigational behaviors generally consist of sequential data and these behaviors may change over time. For this reason, within the scope of the study, it was thought that Sequence-to-Sequence methods would represent user navigational data better than Graph2vec and word embedding methods.

Transformers [41, 42] have revolutionized natural language processing with their attention mechanisms, allowing the model to focus on different input sequence parts when making predictions. In user behavior modeling, transformers capture long-range dependencies and complex patterns in user interactions. Representing user behavior as sequences, and creating embedding vectors with transformers can be useful to predict future actions with high accuracy, and can make them powerful tools for prefetching and caching strategies. In this study, user navigational data was converted into vectors with transformers models and used.

As demand for faster and more efficient data delivery continues to grow, future advancements in prefetching and caching are expected to focus on combining real-time data processing, edge computing, and adaptable machine learning (ML) algorithms. Recent research shows that analyzing user behavior in real-time can make prefetching models more accurate by capturing sudden changes in user actions. Edge computing, which caches data closer to the user, can also help reduce delays. Another promising area is adaptive ML models that learn continuously, adjusting to new patterns over time. These models improve their predictions as they adapt to complex and changing user needs. Together, these trends suggest a shift towards intelligent, distributed caching systems that can quickly adapt to the specific demands of applications and user preferences [8, 9, 43, 44].

Recent studies have explored the integration of computational grids, geographical information systems, and web services to manage and process large-scale data efficiently. For instance, projects such as iSERVO have demonstrated the potential of combining GIS with computational environments to analyze and manage complex geophysical data and simulations effectively [45, 46]. Similarly, initiatives like QuakeSim have leveraged collaborative grid services and web-based portals to enhance data management, user interaction, and computational material science tasks [47]. In the context of software systems, research has also examined structural code clone detection and user interface testing for large-scale graph data, showcasing advanced approaches for handling complex patterns in big data applications [48, 49]. While these works focus on utilizing computational grids and web services for data-intensive applications, the current study diverges by addressing navigational behavior modeling and predictive prefetching in client-server systems, leveraging advanced machine learning techniques such as transformers, graph-based embeddings, and sequence-to-sequence models.

In the literature, predictive prefetching has been studied by training machine learning models with data where user navigational data is represented by label encoding [4, 8, 9]. However, in this study, graph-based, nlp-based, sequential-based, and transformers-based approaches are used to represent the data.

3 | Problem Definition

Prefetching is a technique aimed at reducing latency in client-server systems by proactively loading data that is likely to be requested by the user. In data-intensive environments, the traditional approach to prefetching involves basic caching and prediction methods. However, these approaches are often limited in their ability to capture and predict complex user behaviors, especially in modern applications that operate under unpredictable network conditions and varying user patterns.

The problem addressed in this study centers on the limitations of existing prefetching methods in capturing intricate navigational patterns within client-server systems. While traditional caching mechanisms are effective to some extent, they often fail to adapt to the dynamic demands of mobile and web environments. Existing prefetching solutions typically utilize basic machine-learning models or heuristic-based methods that do not fully exploit advanced modeling techniques. Specifically, these methods often neglect the potential of deep learning approaches, such as autoencoders and Long Short-Term Memory (LSTM) networks, as well as graph-based encoding techniques that could provide more nuanced insights into user navigational patterns.

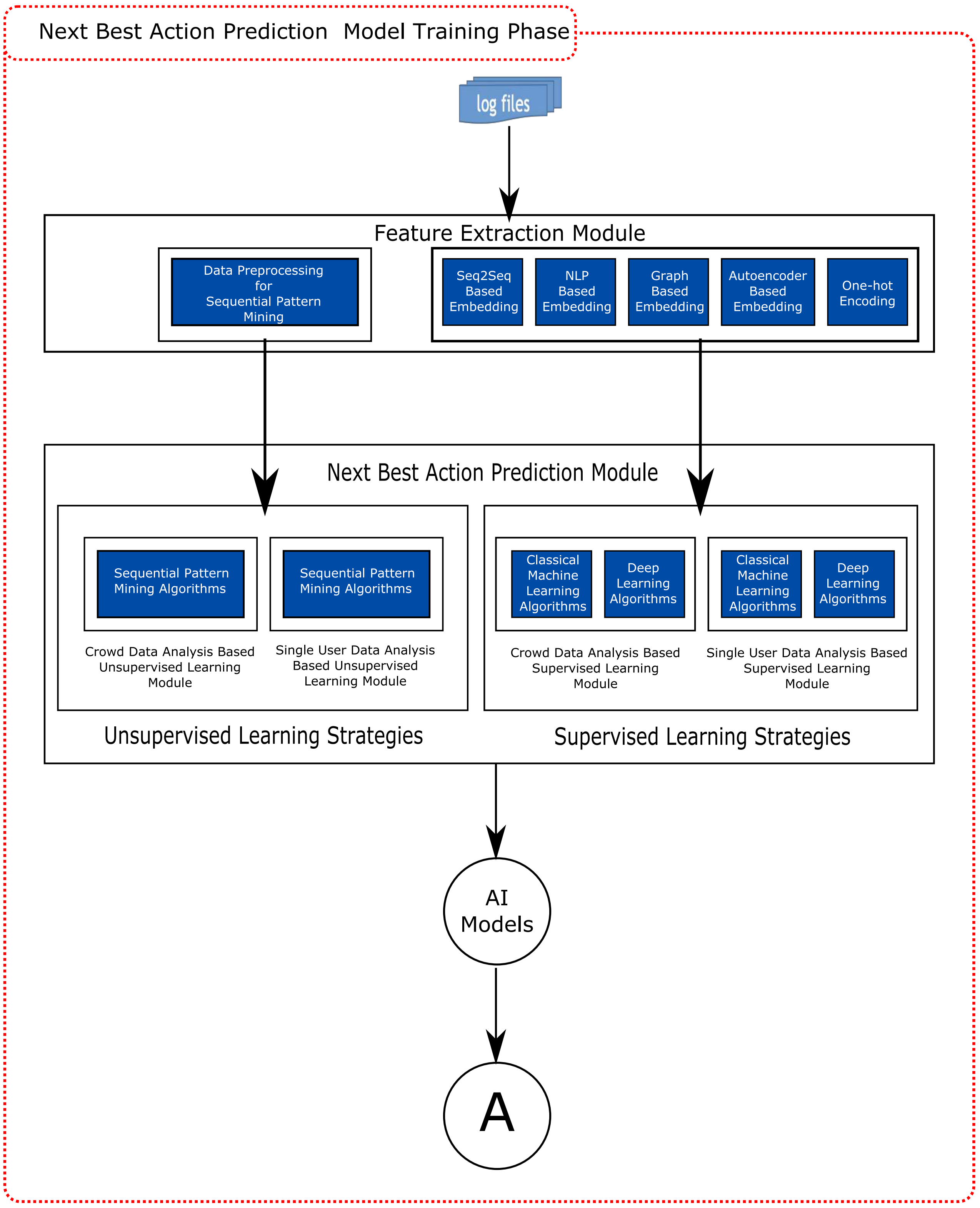
Furthermore, current literature tends to focus on either user-specific data or aggregated data when training predictive models for prefetching, but rarely both. This limitation restricts the effectiveness of prefetching models, as they fail to leverage the full spectrum of available data. The need for hybrid systems that combine both supervised and unsupervised learning approaches remains largely unexplored, despite their potential for improved predictive accuracy and system performance. Therefore, this study addresses these gaps by proposing an advanced prefetching methodology that incorporates sophisticated machine learning techniques and a comprehensive data approach, offering a more robust solution for data-intensive environments.

Research Questions: (1) How can advanced modeling techniques such as autoencoders, LSTM, and graph-based encoding approaches effectively capture the complex patterns in users' navigational behavior for prefetching purposes? (2)What should be the ideal embedding dimension to represent user behavior? (3) How does combining user-specific navigational behavior data and aggregated data from all users affect the accuracy and efficiency of supervised learning models for prefetching? (4) How can a hybrid prefetching system that combines supervised and unsupervised learning approaches be designed and implemented to enhance the performance and adaptability of data-intensive client-server systems? (5) What are the practical implications and performance benefits of the proposed intelligent prefetching framework in mobile applications, particularly in terms of reducing latency and enhancing user experience?

4 | Proposed Methodology

This study aims to represent user navigational data with different vector representation methods and to make effective and innovative prefetching decisions using these representations. Within the scope of the study, an architecture is proposed for creating vector representations of user navigational data with different approaches and training predictive prefetching models from these representations. The proposed architecture consists of two phases. The first phase consists of training predictive prefetching models from historical user navigational data. The second phase consists of deploying the models obtained from the first phase and applying prefetching decisions.

1. Next Best Action Prediction Model Training Phase: This phase of the proposed architecture is illustrated in Figure 1. Accordingly, it is assumed that there is historical user navigational data. The historical data is taken into the feature extraction module. The feature extraction module feeds the prediction model training module and trains the next action prediction model for prefetching.



Next Best Action Prediction Model Training Phase
log files
Feature Extraction Module
Data Preprocessing for Sequential Pattern Mining
Seq2Seq Based Embedding
NLP Based Embedding
Graph Based Embedding
Autoencoder Based Embedding
One-hot Encoding
Next Best Action Prediction Module
Sequential Pattern Mining Algorithms
Sequential Pattern Mining Algorithms
Crowd Data Analysis Based Unsupervised Learning Module
Single User Data Analysis Based Unsupervised Learning Module
Classical Machine Learning Algorithms
Deep Learning Algorithms
Classical Machine Learning Algorithms
Deep Learning Algorithms
Crowd Data Analysis Based Supervised Learning Module
Single User Data Analysis Based Supervised Learning Module
Unsupervised Learning Strategies
Supervised Learning Strategies
AI Models
A

FIGURE 1 | Model training phase.

Feature Extraction Module: This section outlines the preprocessing and embedding techniques used to enhance the representation of user navigation data for predictive prefetching in client-server architectures. These techniques are crucial for effectively modeling and anticipating user behavior, thereby improving the efficiency of data retrieval and system performance. The preprocessing module includes feature extraction processes tailored for training both unsupervised and supervised methods. Specifically, it prepares user navigation data for sequential pattern mining, an unsupervised method that is well-suited for handling large datasets and can be easily integrated into big data tools. Our previous work has demonstrated the effectiveness of sequential pattern mining algorithms in web and mobile preloading contexts using user navigation data [8, 9]. Building on this foundation, we have incorporated this method into the current architecture. In addition, we have previously employed natural language processing (NLP)-based techniques to represent user navigation data, using these representations to enhance predictive prefetching accuracy [7]. In the context of this study, we recommend creating vector representations of user navigational data using NLP-based methods to capture the underlying patterns and relationships more effectively. The proposed architecture employs a variety of embedding techniques, including sequential-to-sequential methods, graph-based embedding methods, and autoencoder-based embedding methods, to more accurately represent user activities. These diverse embedding approaches are integral to our architecture, as they allow for the creation of rich and nuanced representations of user navigation data. Sequential-to-sequential methods capture temporal dependencies and patterns in user actions by treating the sequence of events as input and predicting future events in the sequence. These methods are particularly effective for tasks where the order of user interactions plays a critical role, such as predicting the next page a user will visit. Graph-based embedding methods focus on representing the relationships between different user actions as a graph, where nodes represent the actions and edges represent the transitions between them. This approach is well-suited for capturing complex interactions and dependencies within the data, making it useful for tasks where the network of user behavior needs to be understood and predicted. Autoencoder-based embedding techniques decrease the dimensionality of user navigation data by producing a compressed representation that preserves the most important features. This method is especially advantageous when working with data that has many dimensions, as it can be helpful in finding latent patterns and structures that may not be easily visible. These different embedding methods each offer unique strengths, enabling the architecture to capture various aspects of user behavior. Sequential-to-sequential methods are adept at modeling the order and timing of user actions, graph-based embeddings excel at capturing relational structures, and autoencoder-based methods are powerful for reducing complexity and highlighting essential features.

In this study, the embedding methods generate an embedding output for each user action provided as input. The output size is customizable and can be specified by the user. Accordingly, within the proposed architecture, the model produces an embedding of the designated size for each user action input. When multiple user actions are input, the model calculates the average of the resulting embeddings, thereby creating a consolidated output that represents the combined actions.

The data generated by the Feature Extraction Module is then used to train the predictive models. This data encompasses all users with previous activity, ensuring that the models are trained on comprehensive and representative datasets. The architecture is also designed to support the development of hybrid models that combine both unsupervised and supervised methods, further enhancing the robustness and flexibility of the predictive prefetching system. Moreover, by utilizing these complex placement techniques, the architecture can adapt to different types of user behavior and becomes more efficient in environments with complex user interactions. Correctly predicting and prefetching data is essential for optimizing performance in client-server systems. This capability may significantly reduce latency while improving user experience.

Next Best Action Prediction Model Training Module: In this module, both unsupervised and supervised machine learning models are trained using the collective data from all users. If an individual user has sufficient data, separate unsupervised and supervised models are also trained specifically for that user. Sequential pattern mining techniques are employed for unsupervised training, where the derived rules function as models. For supervised training, we utilize classical machine learning algorithms such as Random Forest, Decision Tree, and K-Nearest Neighbors, along with deep learning algorithms like LSTM and Bi-LSTM. These algorithms are applied to all generated embeddings to build robust and accurate models. Each model is tailored to capture different aspects of user behavior, improving the system´s ability to predict future actions. After the models are trained, they are exported and integrated into the Next Best Action Prefetching Module, where they are used to make real-time predictions. This process ensures that the system remains responsive and efficient, adapting to both collective and individual user behaviors. The inclusion of both classical machine learning and deep learning methods allows for a comprehensive approach to model training. Classical algorithms establish a strong base, whereas deep learning models offer the ability to capture complex patterns in user data. By integrating these methodologies, the system achieves an optimal balance between precision and computational efficacy, enabling it suitable for real-time implementations in client-server systems.

2. Client-Server Architecture with Caching and Prefetching Strategies Phase: The proposed architecture, illustrated in Figure 2, optimizes the performance of a client-server system by incorporating caching and prefetching strategies. In a typical client-server setup, user requests are processed by the server, often leading to slow response times, especially when accessing data from the database. To address this issue, our architecture introduces a cache server between the client and the main server. This cache server reduces the load on the primary server and ensures faster responses to user requests. The architecture includes a Prefetching Module and a Cache Manager. The Prefetching Module uses models trained on historical user navigation data to predict future user requests. These models are periodically updated and deployed to the Next Best Action Prediction Module. The Cache Manager, responsible for managing the cache, uses these predictions to preemptively load data into the cache based on a cache replacement policy. When a user makes a request, the system first checks the cache. If the requested data is available in the cache, it is immediately returned to the user, ensuring a fast response. If the data is not in the cache, the request is forwarded to the main server, where the Cache Manager retrieves the data from the database and sends it to the user. Simultaneously, the system predicts the user's next request using the Next Best Action Prediction Module and preloads the anticipated data into the cache. This process repeats continuously, allowing the system to adapt to user behavior and maintain high performance. The design of this architecture effectively balances the load between the cache server and the main server, reducing latency and improving the overall user

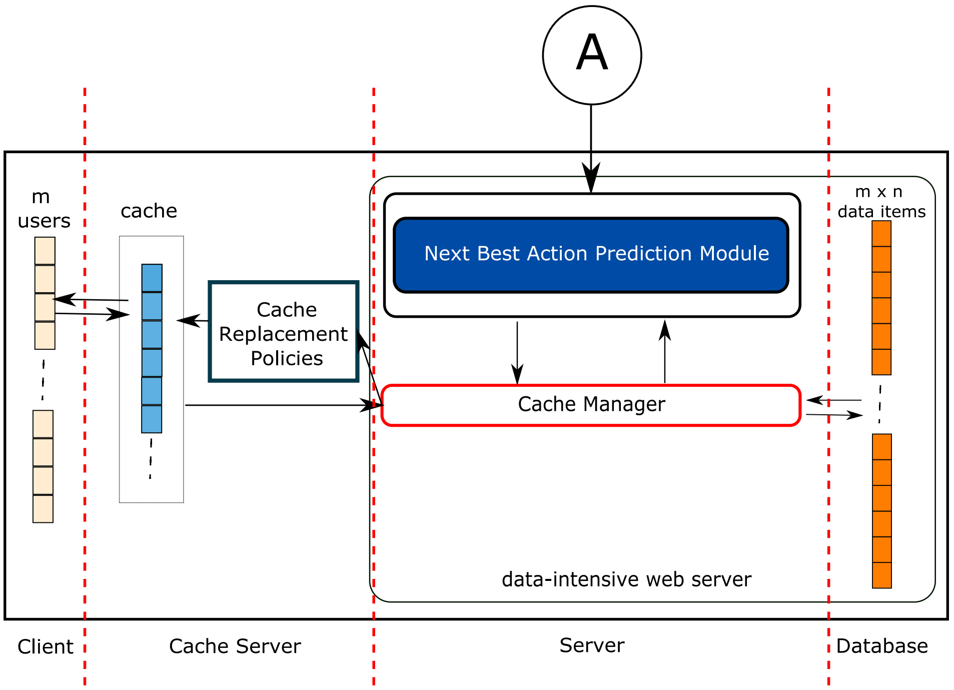

A
m users
cache
Cache Replacement Policies
Next Best Action Prediction Module
Cache Manager
m x n data items
data-intensive web server
Client
Cache Server
Server
Database

FIGURE 2 | Smart prefetching phase.

experience in data-intensive environments. The integration of predictive prefetching and efficient cache management ensures that the system remains responsive and scalable, even as the volume of user requests increases.

5 | Dataset and Prototype Implementation Details

5.1 | Dataset

In our study, we utilized a dataset generated from end-user interactions with a mobile application of a coffee chain store [50]. This dataset includes web sessions recorded in 30-min intervals over 10 days. The pages in the mobile application are personalized for each user, resulting in varied content. For example, users X and Y might see different content when visiting the homepage. The dataset consists of 96,200 unique users and 46 distinct pages, leading to a total of $96,200 \times 46$ unique content instances.

The dataset comprises sessions from users, with each session recorded on a single line using a data representation format derived from the open-source SPMF dataset [51]. An example dataset snippet is represented in Table 1. The IDs corresponding to the pages are listed in Table 2. In this study, each user action is referred to as an event. While interacting with the system, a user may perform a series of events, one after another, which is defined as a sequence. The sequence represents the order of actions taken by the user during their session on the system. An example of such a sequence is shown in the green dotted rectangle in Figure 3. In Table 1, the first 5 rows of the dataset are presented in this table. Each activity log in the dataset is separated by a delimiter of −1. The delimiter of −2 represents the end of the session. Each activity log consists of 14 characters. The first 7 characters represent the user identifier, the next 3 characters represent the data identifier, and the last 4 characters represent the day of the week and the hour of the day, respectively. In this study, time information is not used.

It is important to note that some pages may be visited multiple times during a session. However, for the purpose of making prefetching decisions, we argue that revisited pages do not need to be considered since the necessary data for these revisits has already been retrieved from the database during the initial visit. By focusing on the data from users' initial visits within each session, we were able to extract sequential events, thereby reducing noise from repeated activities. Consequently, a refined dataset was utilized for this study. The refined dataset was subjected to statistical analysis. Figure 4 displays the distribution of sessions per user. The average number of sessions per user is 6.63, while the median is 2. Over 350 users have interacted with the application only once, representing the minimum session count. Conversely, some users have accessed the application up to 71 times across various periods, marking the maximum number of sessions observed in the dataset.

Each session data in the final dataset obtained is divided into 4-dimensional windows. The first 3 data in each window are determined as input, and the last data is determined as label. A suitable dataset is set for predictive prefetching studies by shifting the window.

During a session, users can generate multiple events and subsequently request user-specific data related to these events. Table 2 provides details on the types of data available for user requests. Figure 5 depicts the distribution of events created by users. Analysis of this distribution shows that the most frequently requested data corresponds to page IDs 003, 001, and 002. In contrast, data

TABLE 1 | A snapshot of the dataset.

Session number	Activities in a session
1	00000530010123 −1 00000530020123 −1 00000530000123 −1 00000530020123 −1 00000530030123 −1 00000530040123 −1 00000530110123 −1 00000530010123 −1 00000530070123 −1 00000530010123 −1 00000530050123 −1 00000530130123 −1 00000530140123 −1 00000530120123 −1 00000530080123 −1 00000530090123 −1 00000530000123 −1 00000530010123 −1 00000530020123 −1 00000530020123 −1 00000530030123 −1 00000530050123 −1 00000530040123 −1 −2
2	00000190000123 −1 00000190010123 −1 00000190020123 −1 00000190000123 −1 00000190020123 −1 00000190030123 −1 00000190000123 −1 00000190010123 −1 00000190020123 −1 00000190020123 −1 00000190030123 −1 00000190040123 −1 00000190010123 −1 00000190070123 −1 00000190010123 −1 00000190130123 −1 00000190140123 −1 00000190050123 −1 00000190120123 −1 00000190110123 −1 00000190000123 −1 00000190010123 −1 00000190020123 −1 00000190020123 −1 00000190030123 −1 00000190050123 −1 00000190040123 −1 00000190000123 −1 00000190110123 −1 00000190010123 −1 00000190000123 −1 00000190020123 −1 00000190020123 −1 00000190030123 −1 00000190040123 −1 00000190050123 −1 −2
3	00000160010123 −1 00000160010123 −1 −2
4	00001170010200 −1 −2
5	00000350010123 −1 00000350070123 −1 00000350110123 −1 −2

associated with page IDs 006, 015, and 016 are requested less often.

Table 3 offers a detailed summary of the dataset utilized in our study. Data was collected over a 10-day period and encompasses 96,200 customers participating in 296,100 sessions. The dataset reveals significant variation in user engagement: while the minimum number of sessions per customer is one, some customers have engaged in up to 71 sessions. On average, each customer has approximately 6.63 sessions. Regarding the number of unique events per session, the minimum is 1 and the maximum is 11, with an average of 8.61 unique events per session. The median number of events per session is 10.

5.2 | Prototype Implementation Details

This section details the creation of user navigational data representations, the integration of hybrid prefetching models utilizing both unsupervised and supervised methods, and the prototype implementation of the prefetching architecture for client-server systems. The dataset used, as described in Section 5.1, was preprocessed to reflect user activities within each session, making it suitable for both individual and aggregated user analysis.

TABLE 2 | Page IDs and descriptions of the dataset used in the study.

Page IDs	Request	Description
000	GetOrderHistory	Order history
001	CRM->GetCustomer	Customer information
002	CRM->GetBadgeCampaigns	Customer special campaigns
003	AccountController->Profile/ GetInformations	Customer profile information
004	StoresController->FavoritedStores	Favorite store of the customer
005	PointController->GetMySnaps	Customer reward points
007	CampaignController->Campaigns	Public campaigns
008	CRM->CreateQRCode	QR code generation
011	StoresController->Stores	Stores
013	CRM->GetMobileViewCampaigns	Mobile special campaigns
014	CampaignController-> UserCampaigns	Customer special campaigns in mobile
015	ProductController->GetAll	All products
016	ProductController->Categories	All categories

Figure 3 illustrates an overview of the prototype implementation. For the prototype implementation of Seq2Seq embeddings, we employed an LSTM encoder structure [52], which generates a vector of the specified size for each user action input. Autoencoder-based embeddings were generated using the TensorFlow framework. Word2Vec was utilized to create NLP-based embeddings. Graph-based embeddings were constructed using the DeepWalk and Node2Vec methods. We have used Gensim [53] topic modeling library to create Word2vec, Node2vec, and DeepWalk vectors. The hyperparameters used to create the embedding vectors are presented in detail in Table 4. Most of these parameters are default parameters.

To make prefetching decisions with unsupervised methods, we implemented the PrefixSpan algorithm, a Sequential Pattern Mining technique, which can also be executed in distributed environments like Apache Spark. The resulting embedding vectors were then used as inputs for supervised methods, where models such as K-Nearest Neighbors, Decision Tree, Multi-Layer Perceptron, Random Forest, LSTM, and Bi-LSTM were trained on the embedding data. Hyperparameters of the classification algorithms trained to make predictive prefetching decisions from the generated embedding vectors are presented in Table 5. The parameters used are mostly default parameters.

We also developed a demo application to create a simulation of our work. Screenshots of the demo application are shown in Figures 6 and 7. In our application, the user begins by loading a dataset in SPMF format [51]. Next, the test size is specified, and the remaining portion is automatically set as the training size. If the user chooses to use an unsupervised method, they can

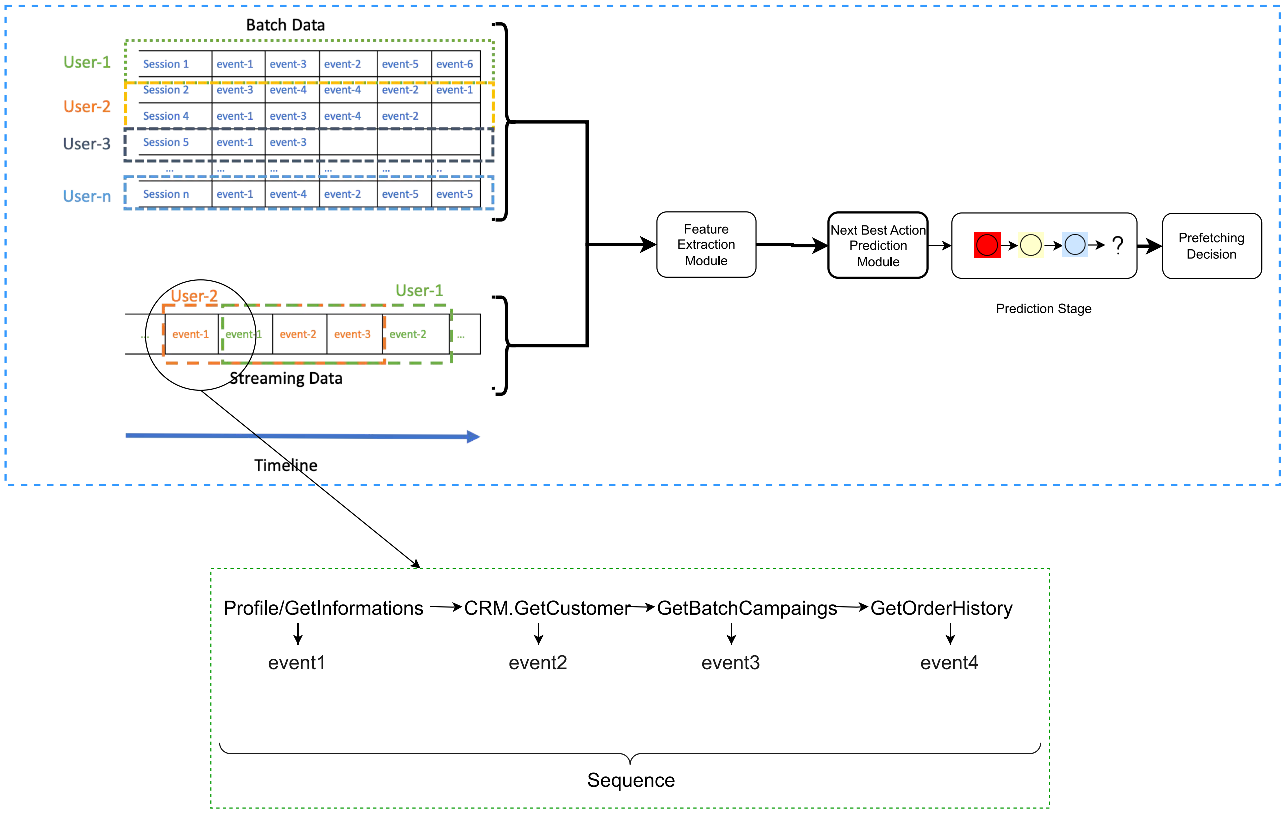

Batch Data
User-1
User-2
User-3
User-n
Session 1 | event-1 | event-3 | event-2 | event-5 | event-6
Session 2 | event-3 | event-4 | event-4 | event-2 | event-1
Session 4 | event-1 | event-3 | event-4 | event-2
Session 5 | event-1 | event-3
Session n | event-1 | event-4 | event-2 | event-5 | event-5
User-2
User-1
event-1 | event-1 | event-2 | event-3 | event-2
Streaming Data
Timeline
Feature Extraction Module
Next Best Action Prediction Module
?
Prefetching Decision
Prediction Stage
Profile/GetInformations → CRM.GetCustomer → GetBatchCampaings → GetOrderHistory
event1
event2
event3
event4
Sequence

FIGURE 3 | The pipeline of the prefetching decision. First, feature extraction is performed from batch data or streaming data, then the Next Best Action Prediction Module predicts the next request, and the data response to the predicted request is cached.

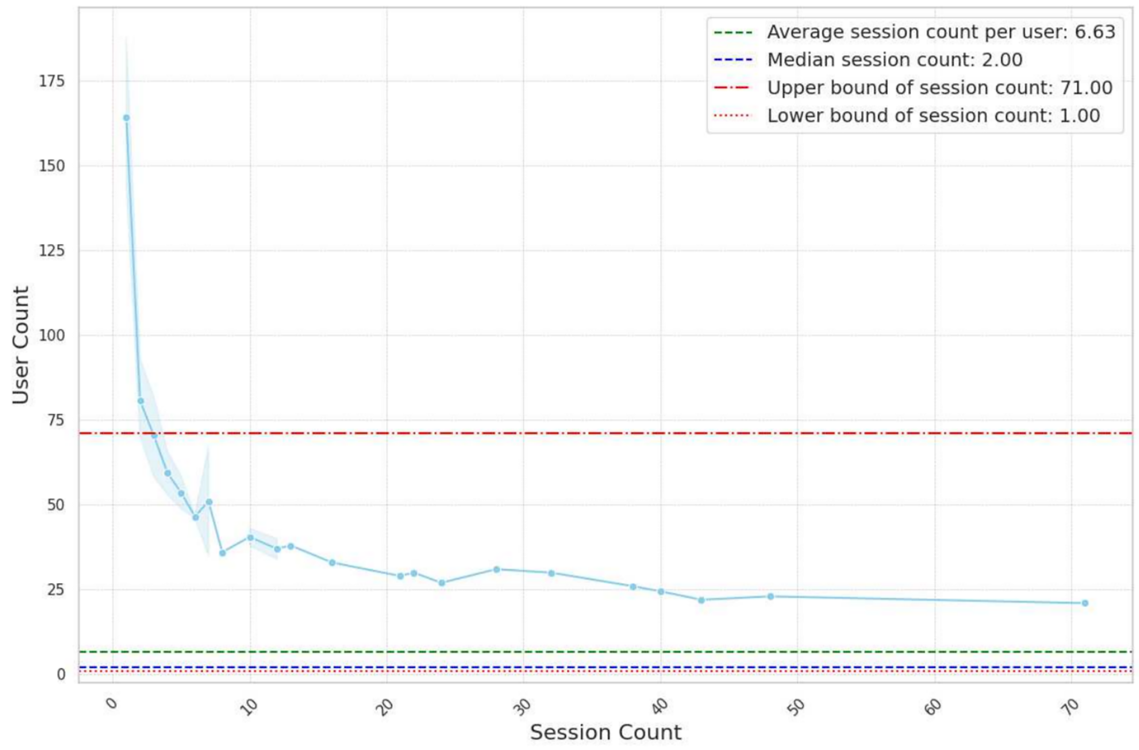


FIGURE 4 | Session distribution: A frequency count of the occurrence of each session across the customers.

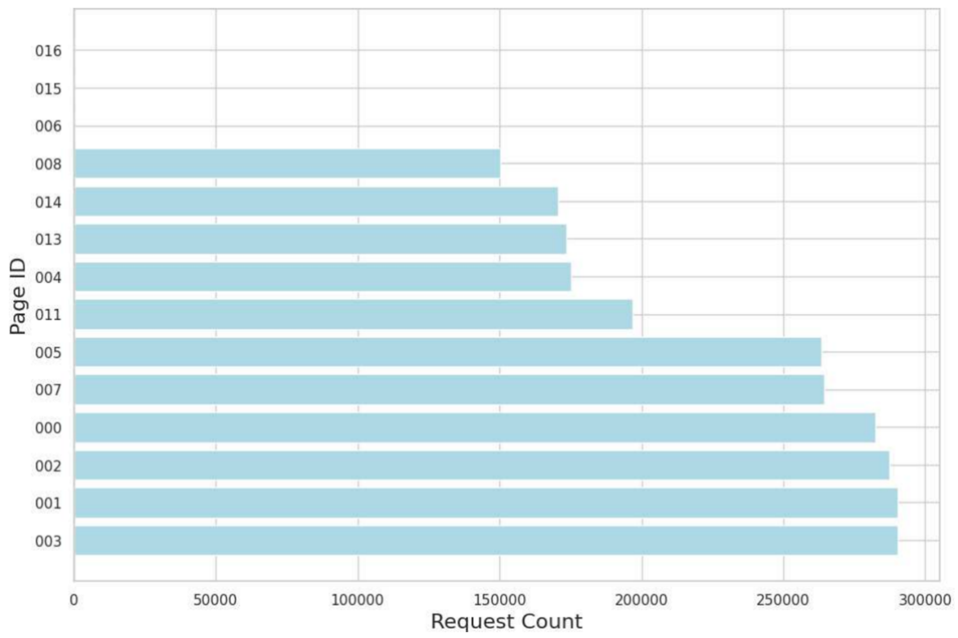


FIGURE 5 | Event distribution: A frequency count of the occurrence of each session across the sessions.

TABLE 3 | Summary of dataset metrics.

Metric	**Value**
Time range of the dataset	14 Days
Number of customers	96,200
Total number of sessions	296,100
Minimum number of sessions per customer	1
Average number of sessions per customer	6.63
Median number of sessions per customer	2
Maximum number of sessions per customer	71
Minimum number of unique events in a session	1
Average number of unique events in a session	8.61
Median number of events in a session	10
Maximum number of events in a session	11

select an option from the “Configuration of Unsupervised Methods” section. Additionally, they can choose one of the prefetching methods we have recommended in our previous study [8]. The application allows the user to select and run multiple methods within the simulation environment.

For supervised methods, the user selects an option from the “Configuration of Supervised Methods” section. In this study, it is necessary to choose at least one of the embedding methods we recommend. When the user clicks the “Train” button, the selected prefetching models are trained using the specified methods, and testing is performed on the test data. The results are then displayed.

If the user wishes to evaluate the prefetching performance within the Web application, they must specify the cache size and click the “Simulation” button. This action fills the cache and calculates the success rate of prefetching based on the test data from the loaded dataset. Currently, the application uses the Least Recently Used (LRU) policy for cache replacement, and it computes cache hits and cache misses as part of the evaluation.

6 | Experimental Study Results

In this section, we present the experimental results from tests conducted using the prototype described in Section 5. Our dataset included three input values and a label for each data point. Each input value was converted into an embedding vector of various sizes using the Node2Vec, DeepWalk, Word2Vec, LSTM Encoders, and Autoencoders methods proposed in this study. We then averaged these vectors to create a mean embedding vector that represents each input sequence, resulting in a unique embedding vector for each input. To determine the most effective embedding vector size, we experimented with vectors of 32, 64, and 128 dimensions. We split the dataset into training and testing sets in an 80/20 ratio, using the training set to train models with K-Nearest Neighbors, Decision Tree, Multi-Layer Perceptron, Random Forest, LSTM, and Bi-LSTM methods. Accuracy assessments across different algorithms revealed that the models performed best with 64-dimensional embeddings, as shown in Table 6. Consequently, we selected 64-dimensional vectors for further testing, thus addressing our research question on identifying the optimal embedding dimension.

For each input in the dataset, 64-element embedding vectors were generated. These vectors were averaged to compute a mean embedding vector representing each input sequence. All other parameters in the embedding calculations were maintained at default settings. The final dataset was divided into training and testing sets with an 80/20 split, ensuring that the test set accurately represented the overall dataset.

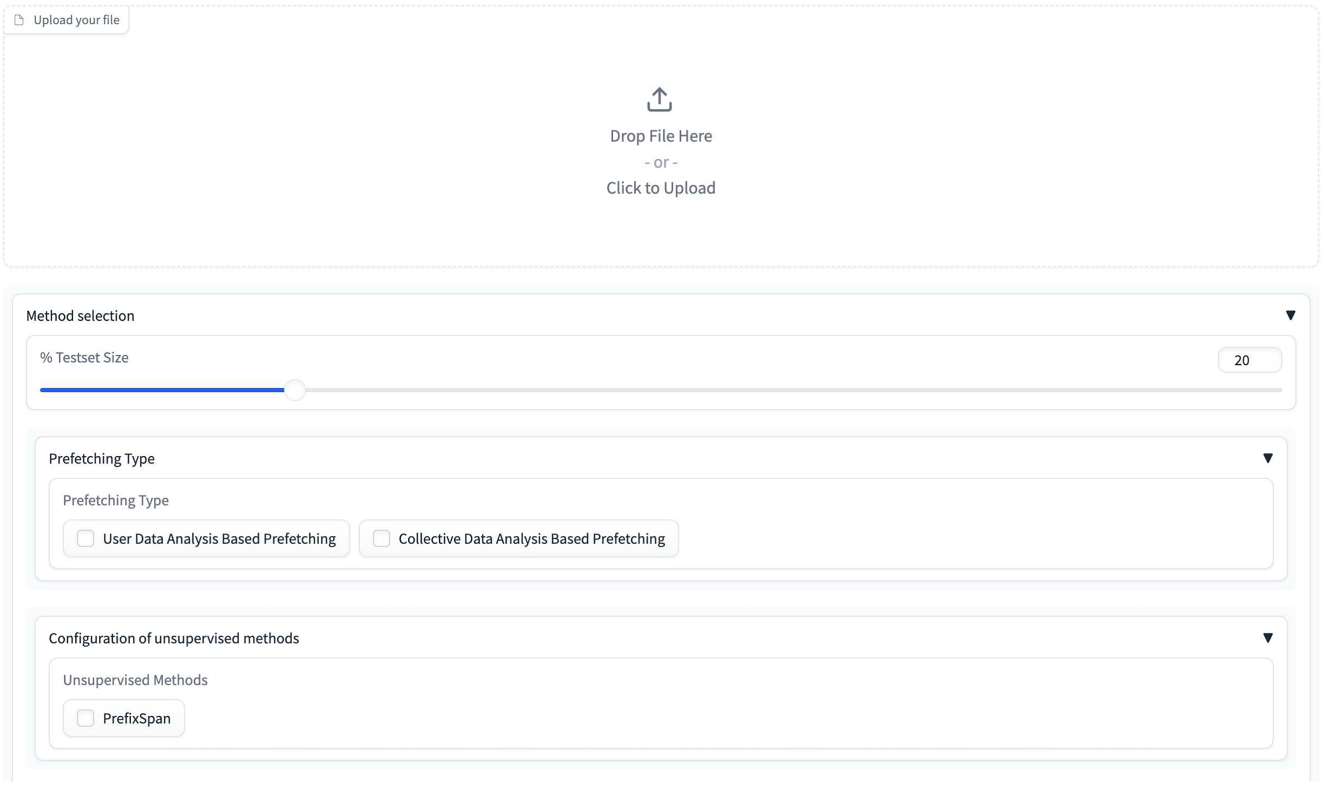
In this experimental study, we found that a significant portion of the server load is generated by the system’s active users. Therefore, a sound business strategy would be to either cache data for

TABLE 4 | Hyper-parameters for embedding algorithms.

Algorithm	Hyperparameter	Value
Word2Vec	Vector_size	64
	Window	5
	Min_count	5
	sg	0
	hs	0
	Negative	5
	Epochs	5
	Alpha	0.025
	Min_alpha	0.0001
	Workers	4
Node2Vec	Walk_length	30
	Num_walks	200
	p	1
	q	1
	Dimensions	64
	Window	10
	Min_count	1
	Batch_size	4
	Iter	1
	Workers	4
Graph2Vec	Max_nodes	100
	Embedding_dim	64
	Batch_size	4
	Num_layers	3
	Num_timesteps	10
	Learning_rate	0.001
	Epochs	100
	Workers	4
Autoencoder	Input_dim	3
	Encoding_dim	64
	Activation	relu
	Loss	mean_squared_error
	Optimizer	adam
	Epochs	100
	Batch_size	32
	Validation_split	0.2
	Dropout	0.0 (no dropout)
	Learning_rate	0.001
	Early_stopping	False
LSTM	units	50
	Activation	relu
	Recurrent_activation	sigmoid
	Use_bias	True
	Dropout	0.0
	Recurrent_dropout	0.0
	Epochs	10
	Batch_size	32
	Learning_rate	0.001
	Optimizer	adam

TABLE 5 | Hyperparameters and values for predictive prefetching classification algorithms.

Algorithm	Hyperparameter	Value
K-Nearest Neighbors (KNN)	n_neighbors	5
	Weights	Uniform
	Algorithm	Auto
	Leaf_size	30
	p	2 (Euclidean distance)
	Metric	Minkowski
Decision tree	Criterion	Gini
	Splitter	Best
	Max_depth	None
	Min_samples_split	2
	Min_samples_leaf	1
	Max_features	None
	Random_state	None
Multi-layer perceptron (MLP)	Hidden_layer_sizes	100
	Activation	Relu
	Solver	Adam
	Alpha	0.0001
	Learning_rate	Constant
	Max_iter	200
	Epochs	200 (uses max_iter as epoch)
	Random_state	None
	Optimizer	Adam
Random forest	n_estimators	100
	Criterion	Gini
	Max_depth	None
	Min_samples_split	2
	Min_samples_leaf	1
	Max_features	sqrt
	Random_state	None
LSTM	Units	50
	Activation	Relu
	Recurrent_activation	Sigmoid
	Use_bias	True
	Dropout	0.0
	Recurrent_dropout	0.0
	Epochs	100
	Optimizer	adam
Bi-LSTM	units	50
	Activation	relu
	Recurrent_activation	sigmoid
	Use_bias	True
	Dropout	0.0
	Recurrent_dropout	0.0
	Merge_mode	concat
	Epochs	100
	Optimizer	Adam


Upload your file

Drop File Here
- or -
Click to Upload

Method selection

% Testset Size 20

Prefetching Type

Prefetching Type

User Data Analysis Based Prefetching

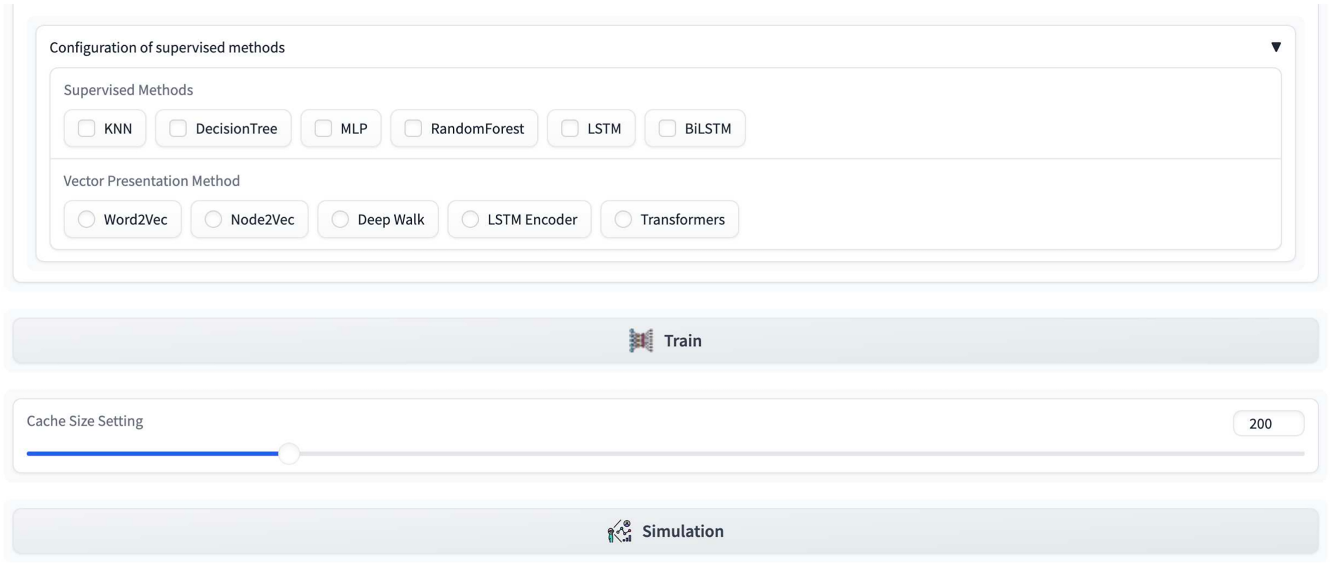
Collective Data Analysis Based Prefetching

Configuration of unsupervised methods

Unsupervised Methods

PrefixSpan

FIGURE 6 | Prototype implementation demo app Screen-1.


Configuration of supervised methods

Supervised Methods

KNN DecisionTree MLP RandomForest LSTM BiLSTM

Vector Presentation Method

Word2Vec Node2Vec Deep Walk LSTM Encoder Transformers

Train

Cache Size Setting 200

Simulation

FIGURE 7 | Prototype implementation demo app Screen-2.

these active users or prioritize improving their overall experience. In prior research, we categorized users as either 'active' or 'cold' based on a predetermined threshold and subsequently developed a next-sequence prediction model tailored specifically for the active user group [9]. However, this classification approach was heavily dependent on the predefined threshold value.

Therefore, in this section, we discuss the details of two experimental studies: (1) Experimental Analysis of Next-Page-Visit Prediction and Prefetching Performance of Machine Learning Models Trained on Aggregated Collective Session Data, (2) Experimental Analysis of Next-Page-Visit Prediction and Prefetching Performance of Machine Learning Models Trained on Individual User Session Data.

Experimental Analysis of Next-Page-Visit Prediction and Prefetching Performance of Machine Learning Models Trained on Aggregated Collective Session Data: Our experimental study evaluated the performance of various machine learning models trained on different embedding

TABLE 6 | Cache miss values for different embedding dimensions.

	Node2Vec			DeepWalk			Word2Vec			LSTMEncoders			AutoEncoders		
	Accuracy			Accuracy			Accuracy			Accuracy			Accuracy		
Dimension	32	64	128	32	64	128	32	64	128	32	64	128	32	64	128
KNN	0,887	0,885	0,883	0,872	0,885	0,883	0,871	0,885	0,876	0,872	0,889	0,885	0,880	0,885	0,879
DecisionTree	0,888	0,885	0,886	0,889	0,885	0,886	0,885	0,885	0,886	0,894	0,895	0,894	0,894	0,894	0,895
MLP	0,885	0,890	0,880	0,888	0,877	0,861	0,882	0,883	0,887	0,880	0,880	0,861	0,867	0,855	0,868
RandomForest	0,885	0,886	0,885	0,889	0,885	0,880	0,885	0,885	0,886	0,895	0,895	0,894	0,890	0,894	0,895
LSTM	0,883	0,890	0,852	0,876	0,881	0,852	0,883	0,884	0,874	0,873	0,889	0,864	0,853	0,859	0,859
BiLSTM	0,883	0,880	0,865	0,877	0,882	0,867	0,870	0,885	0,875	0,882	0,882	0,878	0,843	0,844	0,850

vectors generated using Node2Vec, DeepWalk, Word2Vec, LSTM Encoders, and Autoencoders.

Our findings indicate that among the traditional machine learning models, K-Nearest Neighbors (KNN) and Decision Tree classifiers performed consistently well across all embedding types, with the highest accuracy achieved by the Decision Tree model using LSTM Encoder embeddings (89.5%) and Autoencoder embeddings (89.4%). Similarly, the Random Forest model exhibited high accuracy, particularly with LSTM Encoder (89.5%) and Autoencoder embeddings (89.4%).

For models incorporating deep learning techniques, the Multi-Layer Perceptron (MLP) achieved the best accuracy using Node2Vec embeddings (89.0%) and LSTM Encoder embeddings (88.0%). The LSTM and BiLSTM models also demonstrated competitive performance, with LSTM achieving the highest accuracy using Node2Vec embeddings (89.0%) and LSTM Encoder embeddings (88.9%).

Overall, the results highlight the efficacy of LSTM Encoder embeddings in enhancing prediction accuracy across different machine learning models, with a notable improvement observed in Decision Tree and Random Forest classifiers. This underscores the potential of advanced embedding techniques in improving the performance of predictive models in data-intensive client-server systems.

Our experimental study evaluated the cache hit rates for various machine learning models using different embedding techniques, including Node2Vec, DeepWalk, Word2Vec, LSTM Encoders, and AutoEncoders. We also compared these results with those obtained using the PrefixSpan sequential pattern mining algorithm and a baseline scenario with no prefetching.

The results, which are presented in Figure 8 and Table 7, reveal that the K-Nearest Neighbors (KNN), Decision Tree, and Random Forest models consistently achieved high cache hit rates across all embedding techniques, with KNN and Decision Tree models both reaching 2,123,741 hits using Node2Vec, DeepWalk, and Word2Vec embeddings. Notably, LSTM Encoders slightly improved cache hit rates for these models, with KNN achieving 2,124,449 hits and Decision Tree reaching 2,125,510 hits. AutoEncoder embeddings, while generally effective, resulted in slightly lower hit rates compared to the other embeddings, with BiLSTM achieving 2,116,487 hits.

The MLP and LSTM models also performed well, though their cache hit rates showed some variability depending on the embedding technique used. MLP with Node2Vec achieved 2,124,626 hits, while LSTM with Node2Vec achieved 2,124,626 hits. However, when using AutoEncoders, these models saw a slight decrease in cache hits.

The PrefixSpan algorithm, which uses unsupervised prefetching, achieved 1,981,308 cache hits, indicating its effectiveness, though it fell short of the cache hit rates achieved by the supervised models. The baseline scenario with no prefetching resulted in the lowest cache hit rate of 1,967,154.

Overall, the best result for prefetching decisions is obtained by training tree-based algorithms with embeddings obtained with LSTMEncoder. When this trained model is used, according to the results obtained, 89.5% of cache miss values can be answered from the cache.

Cache misses indicate situations where the requested data is not in the cache and must be retrieved from the server. Lower cache miss rates are indicative of more efficient prefetching and lead to improved system performance. The evaluation of the cache miss graph is the same as the evaluation of the cache hits graph and is presented in Figure 9. The exact cache miss values are displayed in Table 8.

Experimental Analysis of Next-Page-Visit Prediction and Prefetching Performance of Machine Learning Models Trained on Individual User Session Data: In this study, experiments were conducted with varying session counts. As a case study, we selected a user who had utilized the application for 30 sessions and made various requests, training machine learning models specifically for this user. For each session, the dataset's sequences were constructed with three features, and the subsequent request was labeled as the target variable. An embedding dimension of 64 was used to generate word-embedding matrices. The final dataset was split into training and testing sets with an 80/20 ratio, ensuring that the test set was representative of the entire dataset of the user. Models such as K-Nearest Neighbor (KNN), Decision Tree, Multi-Layer Perceptron (MLP), Random

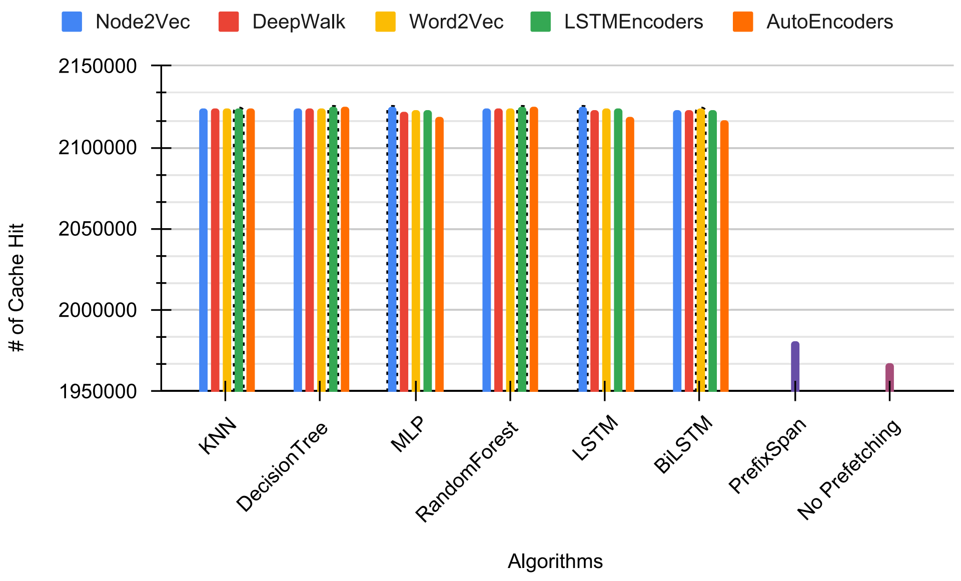


FIGURE 8 | Cache hits for machine learning models trained on aggregated session data with 64-dimensional embedding vectors.

TABLE 7 | Cache hit counts collected from machine learning models on aggregated session data using 64-dimensional embedding vectors, along with results from PrefixSpan and no-prefetching methods.

	Cache hits				
	Node2Vec	**DeepWalk**	**Word2Vec**	**LSTMEncoders**	**AutoEncoders**
KNN	2,123,741	2,123,741	2,123,741	2,124,449	2,123,741
AdaBoost	2,097,378	2066237	2,066,237	2,104,455	2,068,006
DecisionTree	2,123,741	2,123,741	2,123,741	2,125,510	2,125,333
MLP	2,124,626	2,122,325	2,123,387	2,122,856	2,118,433
RandomForest	2,123,918	2123741	2,123,741	2,125,510	2,125,333
LSTM	2,124,626	2,123,033	2,123,564	2,124,449	2,119,141
BiLSTM	2,122,856	2,123,210	2,123,741	2,123,210	2,116,487
	Cache hits				
PrefixSpan	1,981,308				
No prefetching	1,967,154				

Forest, LSTM, and Bi-LSTM were trained using the training set, with all algorithms run using their default parameters.

The Figures 10 and 11 provide a comparison of cache hits and misses across various machine learning models when using different embedding vector datasets, including Node2Vec, DeepWalk, Word2Vec, LSTM Encoders, and AutoEncoders. Cache misses represent instances where the requested data was not found in the cache, necessitating retrieval from a slower storage medium.

Notably, the KNN, DecisionTree, MLP, RandomForest, LSTM, and BiLSTM models consistently exhibit minimal cache misses (1 miss) across all embedding methods, indicating high efficiency in prefetching and data retrieval for these models.

Figure 11 also highlights the performance of the PrefixSpan algorithm, which records 8 cache misses, and the baseline scenario with no prefetching, where cache misses reach 11. This emphasizes the benefits of using embedding vector datasets and machine learning models for reducing cache misses and improving system performance.

These results underscore the importance of using advanced embedding techniques and supervised learning models for prefetching decisions in client-server systems. The improvements in cache hit rates, particularly with the use of LSTM Encoders and AutoEncoders, highlight the potential of these methods to enhance system performance and reduce latency for end-users.

Cost/Benefit Analysis on Aggregated Collective User Session Data: Within the scope of the publication, cache miss has been significantly reduced by representing user behaviors with embedding vectors consisting of 64 dimensions and by performing next action prefetching. Table 9 presents the decrease rate in cache miss values with these methods. According to these results,

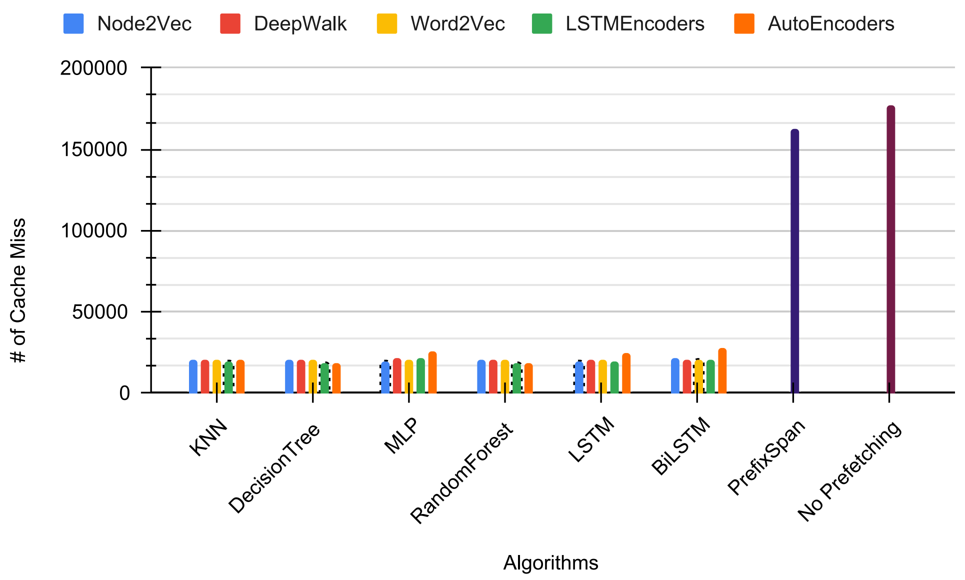


FIGURE 9 | Cache misses of the machine learning models trained on aggregated session data.

TABLE 8 | Cache miss counts recorded with machine learning models on aggregated session data with 64-dimensional embedding vectors, along with results from PrefixSpan and no-prefetching methods.

	Cache misses				
	Node2Vec	**DeepWalk**	**Word2Vec**	**LSTMEncoders**	**AutoEncoders**
KNN	20348	20,348	20,348	19,640	20,348
AdaBoost	46,711	77852	77852	39,634	76,083
DecisionTree	20348	20,348	20,348	18,579	18,756
MLP	19463	21764	20702	21233	25656
RandomForest	20,171	20,348	20,348	18,579	18,756
LSTM	19,463	21,056	20,525	19,640	24,948
BiLSTM	21233	20,879	20,348	20,879	27,602
	Cache misses				
PrefixSpan	162,781				
No prefetching	176,935				

the best performance is obtained by representing user behaviors with LSTM Encoder and predicting the action to be taken by the user with RandomForest. Among the methods we proposed, the models trained with RandomForest usually yielded the best results. Consequently, we conducted analyses with RandomForest and embedding vectors to assess the time consumption during execution.

The tests were performed on a system equipped with four Intel Xeon 2.20 GHz vCPUs (2016 model) and 32 GB of RAM. The runtime costs are presented in Table 10. Table 10 summarizes the runtime costs of RandomForest models using different user behavior embedding techniques. Notably, the embedding methods based on Node2vec, DeepWalk, and Word2Vec demonstrate minimal time consumption for vector creation, each taking less than 1 millisecond (ms). In contrast, AutoEncoder and LSTMEncoder models require significantly more time, with both taking 70 ms for vector creation. However, the prediction time remains consistent across all methods at 6.50 ms. Overall, the total time consumption is substantially lower for Node2vec, DeepWalk, and Word2Vec compared to AutoEncoder and LSTMEncoder, which exhibit higher computational costs primarily due to the time required for embedding generation.

6.1 | High-Performance Computing (HPC) and Prefetching for Enhanced System Performance

The core contribution of this study lies in proposing a technology that enhances high-performance computing systems by utilizing advanced prefetching strategies. Prefetching plays a critical role in reducing latency and improving the responsiveness of HPC systems, especially in environments where large-scale data-intensive applications are executed. The prefetching methods we developed are designed to anticipate user data requests

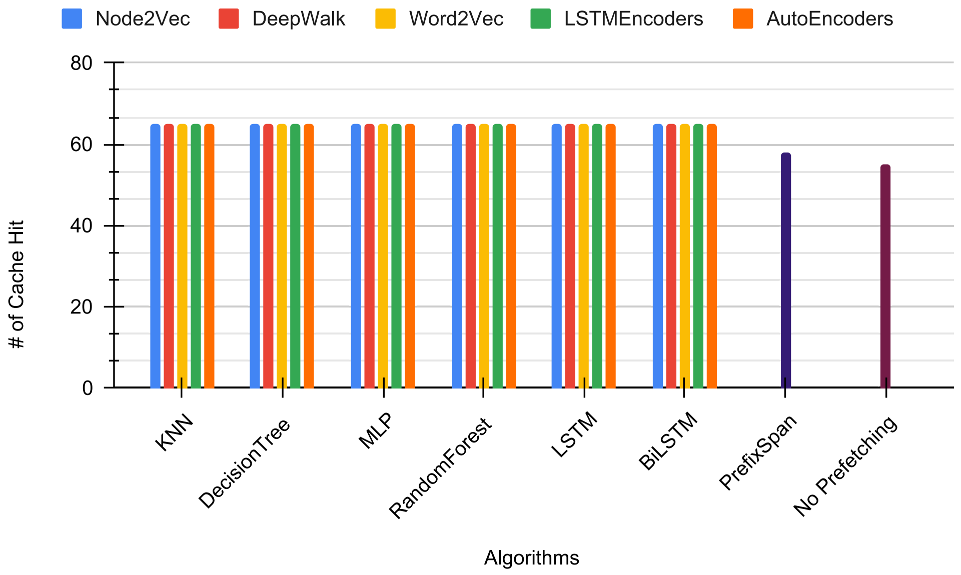


FIGURE 10 | Cache hits of the models trained for the user with ID number 112 with 64-dimensional embedding vectors, who used the application 30 times at different time intervals and performed a total of 66 actions.

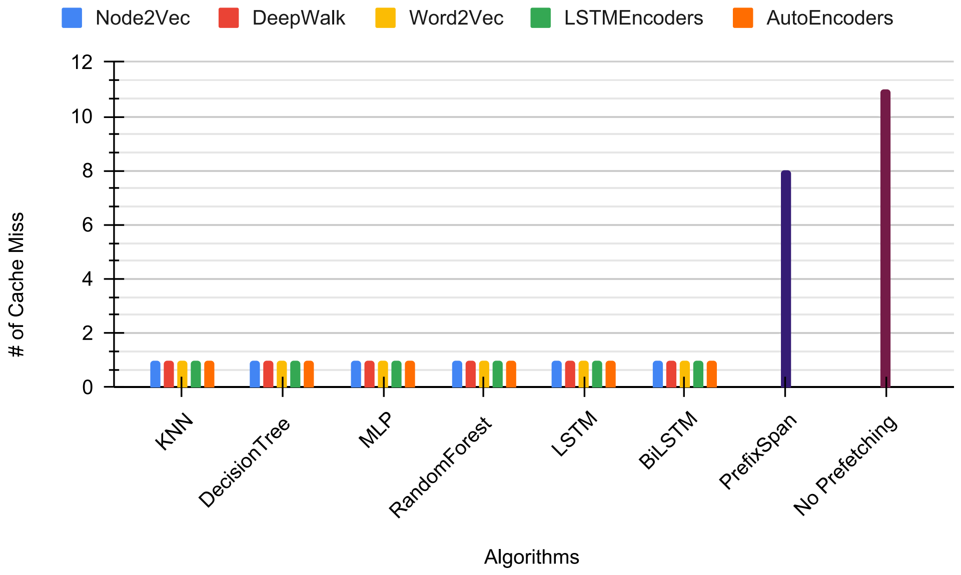


FIGURE 11 | Cache misses of the models trained for the user with ID number 112, who used the application 30 times at different time intervals and performed a total of 66 actions.

and load necessary data into the cache before it is explicitly requested, thereby minimizing the wait time for data retrieval and optimizing system performance.

In HPC systems, where processing large datasets and running complex simulations can be resource-intensive and time-consuming, the integration of intelligent caching and prefetching mechanisms is essential. The proposed framework not only improves the efficiency of data access but also significantly enhances the overall computational throughput. By predicting and preloading data, the system reduces the computational overhead associated with frequent data retrievals, allowing HPC resources to be utilized more effectively for processing tasks rather than data management.

The prefetching system developed in this study can be particularly valuable for HPC environments that handle intensive applications such as simulations, big data analytics, and machine learning model training. By ensuring that the data required for these processes is readily available in the cache, the system accelerates computational tasks, leading to substantial improvements in runtime and resource efficiency. This approach allows HPC systems to better cope with the demands of modern applications, which often require real-time data processing and responsiveness.

Furthermore, the prefetching techniques presented in this study are scalable and can be adapted to increasingly complex and data-intensive HPC applications. As the scale of computational

TABLE 9 | Reduction in cache miss rate by predicting next user actions based on aggregated user behavior embeddings, along with reduction rate from PrefixSpan and no-prefetching methods.

Decrease Rate of Cache Misses (Percentage)					
	Node2Vec	**DeepWalk**	**Word2Vec**	**LSTMEncoders**	**AutoEncoders**
KNN	0,885	0,885	0,885	0,889	0,885
DecisionTree	0,885	0,885	0,885	0,895	0,894
MLP	0,890	0,877	0,883	0,880	0,855
RandomForest	0,886	0,885	0,885	0,895	0,894
LSTM	0,890	0,881	0,884	0,889	0,859
BiLSTM	0,880	0,882	0,885	0,882	0,844
	Decrease rate				
PrefixSpan	0,080				

TABLE 10 | Runtime costs of RandomForest models using various user behavior embeddings, measured on a system with 4 Intel Xeon 2.20 GHz vCPUs (2016 model) and 32 GB RAM.

	Vector creation (ms)	**Prediction (ms)**	**Total time consumption (ms)**
Node2vec	0,36	6,50	6,86
DeepWalk	0,39	6,50	6,89
Word2Vec	0,30	6,50	6,80
AutoEncoder	70,00	6,50	76,50
LSTMEncoder	70,00	6,50	76,50

tasks grows, the ability to predict and efficiently manage data access through prefetching becomes even more critical, making this technology a robust solution for enhancing the performance of high-performance computing systems.

This study was conducted in a standard computational environment, not directly within an HPC infrastructure. However, the proposed prefetching strategies are designed to address challenges that are particularly relevant to HPC environments, such as reducing latency and enhancing throughput in data-intensive applications. The scalability and adaptability of the proposed framework make it suitable for integration into HPC systems, where it can contribute to optimizing computational workflows and managing large-scale data efficiently. Future studies will focus on implementing and testing these strategies directly within HPC environments to validate their applicability and performance in such scenarios.

6.2 | Limitations of the Study and Mitigation Strategies

In any scientific research, it is important to acknowledge potential limitations to ensure transparency. In this study, one limitation is the reliance on prefetching strategies that are heavily dependent on the accuracy of predictive models. While the machine learning models we developed demonstrated strong performance in predicting user behavior, these models are trained on historical data, which may not always perfectly reflect real-time user interactions, especially in dynamic or unpredictable environments. To mitigate this limitation, we incorporated various embedding techniques (such as sequence-to-sequence and autoencoder models) to capture complex user behaviors more effectively. These advanced models help reduce the impact of variations in user behavior, increasing the accuracy of the predictions over time.

Another limitation is related to the computational resources required for running intensive prefetching strategies, especially when dealing with large datasets and complex models. The need for high-performance computing resources is essential to handle the computational complexity of deep learning-based approaches and large-scale simulations. In response to this, we designed our models to be scalable and adaptable to different computational environments. We have also employed hybrid learning approaches, combining both unsupervised and supervised learning methods to optimize computational efficiency. This allows the prefetching models to be deployed effectively, even in environments where HPC resources are limited, by adjusting model complexity based on the available computational power.

7 | Conclusion and Future Work

In this study, we tackled the key issue of enhancing data delivery performance in data-intensive client-server systems by creating an advanced prefetching framework. We identified the shortcomings of traditional caching systems and noted the limitations of existing prefetching methods, especially the absence of advanced modeling techniques and hybrid learning approaches. Our research introduced a comprehensive framework that utilizes graph-based, autoencoder-based, and LSTM-based embedding techniques. It combines both supervised and unsupervised learning methods to improve the efficiency and adaptability of prefetching mechanisms. In the existing literature, there appears to be a significant gap in studies that utilize vector embedding methods to model user behavior for making prefetching decisions. To the best of our knowledge, our work stands alone in this domain, as no other research has been identified that explores this intersection of user behavior modeling and prefetching strategies. This lack of state-of-the-art studies highlights the novelty and relevance of our research. By leveraging user behavior embeddings, our approach not only enhances the predictive capabilities of prefetching mechanisms but also contributes to a deeper understanding of user interactions within client-server systems. As such, our findings have the potential to pave the way for future investigations in this field, underscoring the importance of incorporating advanced modeling techniques in optimizing prefetching performance.

Our experimental study showed that using graph-based, autoencoder-based, and LSTM-based embedding methods effectively models users' navigational behavior. The results indicated that combining these advanced embedding techniques with supervised learning AI models greatly enhanced the accuracy and efficiency of prefetching predictions. The proposed software architecture successfully integrated these advanced machine-learning techniques into existing systems, offering a scalable and adaptable solution for data-intensive client-server environments. These findings confirm that our framework can reduce latency and improve user experience, particularly in mobile applications.

This research makes several key contributions. By integrating advanced embedding techniques, we achieved a more detailed and accurate representation of user behavior, which improved prefetching accuracy. In the scope of the study, an ideal embedding size was determined using a single data set. However, the ideal embedding size may vary depending on the characteristics of the data. The use of both user-specific and aggregated data strengthened the reliability of our predictions. Moreover, our hybrid prefetching system, which combines supervised and unsupervised learning methods, significantly enhanced performance and adaptability. The proposed software architecture supports the smooth implementation of these advanced techniques, establishing a new standard for future research and development in prefetching.

Future research could focus on incorporating real-time data streams to boost the responsiveness and accuracy of the prefetching mechanism. Exploring other advanced machine learning techniques, such as reinforcement learning, may also yield further enhancements. Expanding the framework to accommodate a broader range of applications and environments, including those with complex data structures and greater variability in user behavior, would be beneficial. Additionally, conducting user studies to evaluate the practical impact on user satisfaction and system performance in real-world scenarios would offer valuable insights into the benefits and potential areas for improvement of the proposed framework.

Data Availability Statement

The data that support the findings of this study are openly available in Clickstreams of A Coffee Shop Mobile Applications at https://www.kaggle.com/datasets/tolgabuyuktanir/clickstreams-of-a-coffee-shop-mobile-applications, reference number 50.

References

1. M. Kleppmann, "Designing Data-Intensive Applications: The Big Ideas Behind re Liable," in *Scalable, and Maintainable Systems-O'Reilly* (Sebastopol, CA: O'Reilly Media Inc., 2017).

2. H. B. Abdalla, "A Brief Survey on Big Data: Technologies, Terminologies and Data-Intensive Applications," *Journal of Big Data* 9, no. 1 (2022): 107.

3. L. Benova and L. Hudec, "Comprehensive Analysis and Evaluation of Anomalous User Activity in Web Server Logs," *Sensors* 24, no. 3 (2024): 746.

4. T. Buyuktanir and M. S. Aktas, "Mobile Prefetching and Web Prefetching: A Systematic Literature Review, Computational Science and Its Applications – ICCSA 2022 Workshops," in *Computational Science and Its Applications – ICCSA 2022 Workshops* (Cham: Springer International Publishing, 2022), 75–89.

5. J. Wang, R. Panda, and L. K. John, "Prefetching for Cloud Workloads: An Analysis Based on Address Patterns," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (2017), 163–172, https://doi.org/10.1109/ISPASS.2017.7975288.

6. M. Hort, M. Kechagia, F. Sarro, and M. Harman, "A Survey of Performance Optimization for Mobile Applications," *IEEE Transactions on Software Engineering* 48, no. 8 (2021): 2879–2904.

7. T. Buyuktanir and M. S. Aktas, "Predictive Prefetching in Client-Server Systems: A Navigational Behavior Modeling Approach," *International Journal of Software Engineering and Knowledge Engineering* 33, no. 11 (2024): 1807–1830, https://doi.org/10.1142/S0218194024500384.

8. T. Buyuktanir, I. Sigirci, and M. Aktas, "Enhancing Accessibility to Data in Data-Intensive Web Applications by Using Intelligent Web Prefetching Methodologies," *International Journal of Software Engineering and Knowledge Engineering* 33, no. 9 (2023): 1405–1438.

9. T. Buyuktanir and M. S. Aktas, "A Deep Learning-Based Prefetching Approach to Enable Scalability for Data-Intensive Applications," in *2022 IEEE International Conference on Big Data (Big Data)* (Osaka, Japan: IEEE, 2022), 2716–2721. https://doi.org/10.1109/BigData55660.2022.10020591.

10. G. Shang, P. Zhe, X. Bin, and S. Yubo, "Secure and Energy Efficient Prefetching Design for Smartphones," in *2016 IEEE International Conference on Communications (ICC)* (Kuala Lumpur, Malaysia: IEEE, 2016), 1–6.

11. E. Olmezogullari and M. S. Aktas, "Pattern2Vec: Representation of Clickstream Data Sequences for Learning User Navigational Behavior," *Concurrency and Computation: Practice and Experience* 34, no. 9 (2022): e6546.

12. E. Olmezogullari and M. S. Aktas, "Representation of Click-Stream Datasequences for Learning User Navigational Behavior by Using Embeddings, 2020 IEEE International Conference on Big Data (Big Data)," in *2020 IEEE International Conference on Big Data (Big Data)* (Atlanta, GA: IEEE, 2020), 3173–3179.

13. L. Boratto, S. Carta, G. Fenu, and R. Saia, "Using Neural Word Embeddings to Model User Behavior and Detect User Segments," *Knowledge-Based Systems* 108 (2016): 5–14.

14. C. Chen, S. Kim, H. Bui, et al., "Predictive Analysis by Leveraging Temporal User Behavior and User Embeddings," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)* (Torino, Italy: ACM, 2018), 2175–2182.

15. M. Pavlovski, J. Gligorijevic, I. Stojkovic, et al., "Time-Aware User Embeddings as a Service," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'20)* (Virtual Event, CA: ACM, 2020), 3194–3202.

16. Y. Zhu, H. Li, Y. Liao, et al., "What to Do Next: Modeling User Behaviors by Time-LSTM," in *Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 17)* (Melbourne, Australia: IJCAI, 2017), 3602–3608.

17. Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to Cache: Machine Learning for Network Edge Caching in the Big Data Era," *IEEE Wireless Communications* 25, no. 3 (2018): 28–35.

18. C. Liu, S. Ding, L. Ye, X. Chen, and W. Zhu, "Cache Replacement Strategy Based on User Behaviour Analysis for a Massive Small File Storage System," in *2022 14th International Conference on Computer and Automation Engineering (ICCAE)* (Brisbane, Australia: IEEE, 2022), 178–183.

19. M. Joo, Y. An, H. Roh, and W. Lee, "Predictive Prefetching Based on User Interaction for Web Applications," *IEEE Communications Letters* 25, no. 3 (2020): 821–824.

20. C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli, "Mining Behavior Models From User-Intensive Web Applications," in *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India: ACM, 2014), 277–287.

21. M. I. Zulfa, R. Hartanto, and A. E. Permanasari, "Caching Strategy for Web Application–a Systematic Literature Review," *International Journal of Web Information Systems* 16, no. 5 (2020): 545–569.

22. L. Karanam, S. Barla, B. Ramamurthy, and D. Weitzel, "ML-Based Adaptive Prefetching and Data Placement for US HEP Systems," *Conference on Computing in High Energy and Nuclear Physics* (2024).

23. Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "A Hierarchical Neural Model of Data Prefetching," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual Event, USA: ACM, 2021), 861–873.

24. W. Ali, S. M. Shamsuddin, A. S. Ismail, et al., "A Survey of Web Caching and Prefetching," *Journal of Advances in Soft Computing and its Applications* 3, no. 1 (2011): 18–44.

25. J. Gomez-Vilardebo, "Fundamental Limits of Caching: Improved Rate-Memory Tradeoff With Coded Prefetching," *IEEE Transactions on Communications* 66, no. 10 (2018): 4488–4497.

26. S. Feng, G. Cong, A. Khan, X. Li, Y. Liu, and Y. M. Chee, "Inf2vec: Latent Representation Model for Social Influence Embedding," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (Paris, France: IEEE, 2018), 941–952.

27. S. Mitrovic and J. De Weerdt, "Dyn2Vec: Exploiting Dynamic Behaviour Using Difference Networks-Based Node Embeddings for Classification, Proceedings of the International Conference on Data Science," in *Proceedings of the International Conference on Data Science* (Las Vegas, NV: CSREA Press, 2019), 194–200.

28. M. Grohe, "word2vec, node2vec, graph2vec, x2vec: Towards a Theory of Vector Embeddings of Structured Data, Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems," in *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Portland, OR: ACM, 2020), 1–16.

29. H. Hakvar, C. Cavuldak, O. Söyler, Y. Karadayi, and M. S. Aktaş, "Time-Sensitive Embedding for Understanding Customer Navigational Behavior in Mobile Banking," in *International Conference on Computing, Intelligence and Data Analytics* (Kocaeli, Turkiye: Springer, 2022), 257–270.

30. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," arXiv preprint, arXiv:1301.3781 (2013).

31. J. Pennington, R. Socher, and C. D. Manning, "Glove: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Doha, Katar: Association for Computational Linguistics, 2014), 1532–1543.

32. R. Esmeli, M. Bader-El-Den, and H. Abdullahi, "Using Word2Vec Recommendation for Improved Purchase Prediction," in *2020 International Joint Conference on Neural Networks (IJCNN)* (Glasgow, UK: IEEE, 2020), 1–8.

33. S. Jarang, D. Joshi, and V. Deshpande, "Behaviour Analysis Using Word Embedding & Machine Learning on Social Media," in *2019 5th International Conference on Computing, Communication, Control and Automation (ICCUBEA)* (Pune, India: IEEE, 2019), 1–6.

34. A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, CA: ACM, 2016), 855–864.

35. B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online Learning of Social Representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY: ACM, 2014), 701–710.

36. Y. Qiu, Y. Gong, and G. Liu, "User Behavior Analysis and Clustering in Peace Elite: Insights and Recommendations," arXiv preprint, arXiv:2407.11772 (2024).

37. A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning Distributed Representations of Graphs," arXiv preprint, arXiv:1707.05005 (2017).

38. L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning Node Representations From Structural Identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, Canada: ACM, 2017), 385–394.

39. A. Graves and J. Schmidhuber, "Framewise Phoneme Classification With Bidirectional LSTM and Other Neural Network Architectures," *Neural Networks* 18, no. 5–6 (2005): 602–610.

40. I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning With Neural Networks," *Advances in Neural Information Processing Systems* 27 (2014): 3104–3112.

41. T. Lin, Y. Wang, X. Liu, and X. Qiu, "A Survey of Transformers," *AI Open* 3 (2022): 111–132.

42. A. Vaswani, "Attention is All You Need," arXiv preprint, arXiv:1706.03762 (2017).

43. C. Li, M. Song, S. Du, X. Wang, M. Zhang, and Y. Luo, "Adaptive Priority-Based Cache Replacement and Prediction-Based Cache Prefetching in Edge Computing Environment," *Journal of Network and Computer Applications* 165 (2020): 102715.

44. F. Zhang, C. Xu, Y. Zhang, et al., "Edgebuffer: Caching and Prefetching Content at the Edge in the Mobilityfirst Future Internet Architecture," in *2015 IEEE 16th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (Boston, MA: IEEE, 2015), 1–9.

45. M. Aktas, G, Aydin, A. Donnellan, et al., "iSERVO: Implementing the International Solid Earth Research Virtual Observatory by Integrating Computational Grid and Geographical Information Web Services," *Computational Earthquake Physics: Simulations, Analysis and Infrastructure* Part II (2007): 2281–2296.

46. G. Aydin, A. Sayar, H. Gadgil, and M. Aktas, "Building and Applying Geographical Information System Grids," *Concurrency and Computation: Practice and Experience* 20, no. 14 (2008): 1653–1695.

47. M. Pierce, G. C. Fox, M. Aktas, et al., "The QuakeSim Project: Web Services for Managing Geophysical Data and Applications," in *Earthquakes: Simulations, Sources and Tsunamis* (Basel, Switzerland: Springer, 2008), 635–651.

48. M. Kapdan, M. Aktas, M. Yigit, "On the Structural Code Clone Detection Problem: A Survey and Software Metric-Based Approach," in *Computational Science and Its Applications–ICCSA 2014: 14th International Conference, Guimarães, Portugal, June 30–July 3, 2014, Proceedings, Part V 14* (Guimarães, Portugal: Springer, 2014), 492–507.

49. Y. Uygun, R. F. Oguz, E. Olmezogullari, and M. S. Aktas, "On the Large-Scale Graph Data Processing for User Interface Testing in Big Data Science Projects," in *2020 IEEE International Conference on Big Data (Big Data)* (Atlanta, GA, USA: EEE, 2020), 2049–2056.

50. T. Buyuktanir, “Clickstreams of A Coffee Shop Mobile Applications” (2022), https://www.kaggle.com/datasets/tolgabuyuktanir/clickstreams-of-a-coffee-shop-mobile-applications.

51. P. Fournier-Viger, J. C. W. Lin, A. Gomariz, et al., “The SPMF Open-Source Data Mining Library Version 2. Machine Learning and Knowledge Discovery in Databases: European Conference,” in *ECML PKDD 2016, Riva del Garda, Italy, September 19–23, 2016, Proceedings, Part III 16* (Riva del Garda, Italy: Springer, 2016), 36–40.

52. Lkulowski, “Lkulowski/LSTM_Encoder_Decoder: Build a LSTM Encoder-Decoder Using Pytorch to Make Sequence-to-Sequence Prediction for Time Series Data,” https://github.com/lkulowski/LSTM_encoder_decoder.

53. R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling With Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (Valletta, Malta: ELRA, 2010), 45–50.