Optimized Cache Placement in Edge Computing Using Machine Learning-Based Predictive Models

1st Yasin Aril Mustofa
Department of Electrical Engineering
Universitas Hasanuddin
Bontomarannu, Gowa, Indonesia
mustofaya22d@student.unhas.ac.id

2nd Syafruddin Syarif
Department of Electrical Engineering
Universitas Hasanuddin
Bontomarannu, Gowa, Indonesia
syafruddin.s@eng.unhas.ac.id
Corresponding Author

3rd Muhammad Niswar
Department of Electrical Engineering
Universitas Hasanuddin
Bontomarannu, Gowa, Indonesia
niswar@unhas.ac.id

***Abstract*—Cache placement in edge computing is crucial in optimizing data retrieval efficiency, particularly in environments characterized by fluctuating content demand and constrained storage capacity. This study proposes a machine learning-driven approach to enhance cache placement decisions, employing Logistic Regression and Support Vector Machines as predictive models. To refine classification accuracy and computational efficiency, the models are optimized through gradient-based techniques such as Stochastic Gradient Descent, Mini-Batch Gradient Descent, Momentum Gradient Descent, Adam, and Newton's Method. The methodology involves processing real-world access data, where key features such as response time, access frequency, and file size inform caching decisions. Experimental evaluations reveal that Newton's Method delivers the highest classification accuracy 97.5% with rapid convergence, while Momentum Gradient Descent achieves the highest cache hit rate 13% at a cache size of 150 MB. Furthermore, the proposed machine learning models significantly surpass traditional caching strategies such as Least Recently Used and Least Frequently Used, achieving up to a 100% improvement in cache hit rate. These findings underscore the effectiveness of adaptive, data-driven caching techniques in enhancing network performance and reducing latency in real-time edge computing environments.**

***Keywords*—Edge Computing, Cache Placement, Machine Learning, Optimization Algorithms, Cache Hit Rate**

I. Introduction

Edge computing has become a pivotal paradigm in modern distributed systems, enabling efficient data processing and storage closer to end users. This architectural shift reduces latency, optimizes bandwidth, and decreases reliance on centralized cloud infrastructures. However, effective cache placement in edge environments remains challenging due to limited storage, fluctuating content popularity, and unpredictable access patterns. Conventional caching methodologies, including Least Recently Used (LRU) and Least Frequently Used (LFU), exhibit a deficiency in adaptation to fluctuating settings, frequently resulting in subpar performance [8], [14]. LRU indiscriminately evicts older content, while LFU struggles with sudden shifts in content demand, resulting in inefficient cache allocations [3].

Machine learning (ML)–based caching approaches have emerged as promising alternatives to address these limitations, leveraging predictive models to optimize cache allocation in real-time [2]. Reinforcement learning (RL) dynamically learns optimal caching policies by interacting with network environments, maximizing cache hit rates, and minimizing latency [10], [4]. Supervised learning techniques, including Support Vector Machines (SVM) and Logistic Regression (LR), utilize historical data to predict content demand and enable proactive cache placement [8], [6]. These models outperform traditional heuristics by identifying complex access patterns and adapting caching strategies accordingly [4], [5], [7].

Furthermore, user mobility and network variability complicate cache placement. Machine learning enhances efficiency through mobility-aware caching that dynamically adjusts to user movement and fluctuating demand. Optimization techniques, such as deep reinforcement learning and transfer learning, improve adaptability and reduce training times [18]. This study evaluates ML-based cache placement, focusing on LR and SVM models optimized with gradient-based techniques to improve cache hit rates and reduce retrieval latency, bridging the gap between traditional and adaptive strategies in edge computing. The key contributions of this study include:

- Cache placement is framed as a binary classification problem using Logistic Regression (LR) and Support Vector Machines (SVM) with advanced optimization techniques.
- Gradient-based optimizations (SGD, MBGD, MGD, Adam, Newton's Method) are evaluated for their effect on model performance.
- Simulations show ML-based caching outperforms heuristics (LRU, LFU) in cache hit rates.

II. Related Work

A. Traditional and Heuristic-Based Cache Placement Strategies

Cache placement in edge computing is essential for minimizing latency and enhancing network efficiency. Conventional methods such as Least Recently Used (LRU) and Least Frequently Used (LFU) are prevalent but exhibit significant shortcomings. LRU removes older content without regard for future demand, whereas LFU preserves frequently accessed data but falters with abrupt changes in content popularity. These constraints may result in inefficiencies, such as the prematurely eviction of high-demand information or the retention of obsolete data. While LRU and LFU improve response

times by caching frequently accessed content near users, they often struggle to adjust to changing user behavior, leading to inefficient cache distribution, heightened network congestion, and prolonged retrieval durations. [13]. Although heuristic-based methods such as probabilistic caching and adaptive eviction policies have been suggested, their capacity to adapt to fluctuating network conditions and user requirements is constrained, impacting overall performance.

B. Machine Learning and Optimization for Adaptive Cache Placement

Machine learning (ML) has arisen as an effective method for enhancing cache placement in edge computing, providing dynamic flexibility to fluctuating network conditions and user access patterns. Supervised learning methodologies such as Support Vector Machines (SVM) and Logistic Regression (LR) utilize previous data to forecast content demand, enhancing cache hit rates and minimizing unwarranted evictions [8], [6]. Reinforcement learning (RL) further enhances cache decisions by continuously learning from user interactions and adjusting placement strategies [10]. Despite these advantages, traditional ML models face challenges regarding convergence speed, accuracy, and computational efficiency. To address this, advanced optimization techniques such as Adam, Momentum Gradient Descent, and Newton's Method have been integrated, significantly improving performance in dynamic environments [13], [17]. Additionally, deep reinforcement learning (DRL) enhances adaptability by predicting content popularity, while transfer learning facilitates deployment across diverse edge environments. Evolutionary algorithms, including genetic algorithms and greedy optimization, further refine cache allocation, ensuring efficient storage utilization and energy conservation [18], [16].

III. Proposed Work

This research proposes a machine learning-based framework for cache placement in edge computing to improve caching efficiency using predictive modeling and optimization. It outlines the dataset, methodology, and performance evaluation framework.

A. Support Vector Machines (SVM) Optimized for Cache Placement

Cache placement can be conceptualized as a binary classification issue, aiming to ascertain whether a data request should be cached $(+1)$ or not (-1). In a dataset of feature vectors $X = [x_1, x_2, ..., x_n]$ denoting attributes like response time, access frequency, and file size, with associated labels $y \in \{-1, 1\}$ signifying cacheability, SVM seeks to determine an optimal hyperplane that maximizes the margin between the two classes.

In this study, the cache placement problem is formulated as a binary classification problem, where a decision is made to either cache a content request (+1) or discard it (-1). The SVM model is trained on a dataset of real-world access patterns, where key features such as response time, access frequency, and file size influence caching decisions. The optimization goal is to maximize cache efficiency by minimizing misclassification errors and adjusting the hyperplane dynamically to changing data distributions.

The optimization challenge for cache placement using SVM is delineated as follows:

$$\min_{w, w_0, \xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i \tag{1}$$

subject to:

$$y_i(w \cdot x_i + w_0) \geq 1 - \xi_i, \quad \forall i = 1, \ldots, N \tag{2}$$

Here, w is the weight vector that characterizes the hyperplane, w_0 represents the bias term that modifies the decision boundary, ξ_i are slack variables that permit misclassification, and C is a regularisation parameter that reconciles margin maximization with error tolerance.

This optimization process, from margin maximization to prediction and evaluation, is illustrated in **Fig. 1**, providing a comprehensive overview of how SVM facilitates effective cache placement decisions.

```
Algorithm 1: Support Vector Machine (SVM) Without Optimization

Input:
    - Feature matrix (X): shape (n_samples, n_features)
    - Target labels (y): shape (n_samples,), with labels ∈ {1,1}
    - Regularization parameter (C): controls trade-off between
margin maximization and error penalty
Output:
    - Final weight vector (w)
    - Final bias (w₀)

Core Process:
    1. Solve the SVM Optimization Problem:
       Minimize the objective function:
         (1/2) * ||w||² + C * Σ ξᵢ
       Subject to:
         yᵢ * (wᵀ · Xᵢ + w₀) ≥ 1 - ξᵢ
         ξᵢ ≥ 0 for all i
    2. Convert to Dual Problem (Lagrangian Dual Form):
       Maximize the Lagrangian:
         L(α) = Σ αᵢ - (1/2) * ΣΣ αᵢ * αj * yᵢ * yj * (Xᵢ · Xj)
       Subject to:
         Σ αᵢ * yᵢ = 0
         0 ≤ αᵢ ≤ C for all i
    3. Compute Model Parameters:
       - Weight vector:
         w = Σ αᵢ * yᵢ * Xᵢ
       - Bias (w₀):
         Select any support vector Xₖ such that 0 < αₖ < C:
             w₀ = yₖ - (wᵀ · Xₖ)
    4. Prediction (Decision Function):
       For each test sample X_test:
         ŷ = sign(wᵀ · X_test + w₀)
Return:
    - Optimized weights (w) and bias (w₀)
```

Fig. 1: Pseudo-code SVM without Optimization

B. Logistic Regression (LR) Optimized for Cache Placement

While SVM focuses on maximizing the margin for classification, Logistic Regression (LR) takes a probabilistic approach to cache placement by predicting the likelihood of a content request using the sigmoid function:

$$P(y = 1 \mid I) = \frac{1}{1 + e^{-(Xw + w_0)}} \tag{3}$$

Model training refines w and w_0 through the minimisation of the log-loss function:

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \quad (4)$$

where N represents the sample size, y_i is the true label, and $\hat{y}_i$ signifies the anticipated probability. This diminishes categorization mistakes, hence improving cache placement precision. The LR process, encompassing initialization to convergence, is illustrated in **Fig. 2**.

```
Algorithm 2: Logistic Regression (LR) Without Optimization

Input:
    - Feature matrix (X): shape (n_samples, n_features)
    - Target labels (y): shape (n_samples,)
    - Learning rate (α)
    - Number of epochs (n_epochs)
    - Weights (w) initialized to 0
    - Bias (w₀) initialized to 0
Output:
    - Final weights (w)
    - Final bias (w₀)

Core Process:
    For each epoch from 1 to n_epochs:
        For each training sample (xᵢ, yᵢ):
            1. Compute linear combination:
                z = wᵀ * xᵢ + w₀
            2. Apply sigmoid activation function:
                ŷ = 1 / (1 + exp(-z))
            3. Calculate error:
                error = ŷ - yᵢ
            4. Update weights and bias:
                w = w - α * (error * xᵢ)
                w₀ = w₀ - α * error
Return:
    - Optimized weights (w) and bias (w₀)
```

Fig. 2: Pseudo-code LR without Optimization

C. Implementation of Optimization Techniques in SVM and LR

Various gradient-based optimization techniques are applied to both SVM and LR to enhance model performance. These methods improve convergence stability and accuracy in predicting cacheable content.

- Stochastic Gradient Descent (SGD)
  SGD updates model parameters iteratively using a single data point per step:

$$w \leftarrow w - \alpha \nabla_w L \quad (5)$$

  For SVM, the gradient is computed using the hinge loss function:

$$L(w) = \sum_{i=1}^{N} \max(0, 1 - y_i(w \cdot x_i + w_0)) \quad (6)$$

  For LR, the gradient is computed from the log-loss function:

$$\nabla_w L = X^T(P - y) \quad (7)$$

  where, $P = \sigma(Xw + w_0)$ represents predicted probabilities.

- Mini-Batch Gradient Descent (MBGD)
  MBGD updates weights using gradients from a small batch:

$$w \leftarrow w - \frac{\alpha}{m}\sum_{i=1}^{m} \nabla_w L \quad (8)$$

  For SVM, the gradient is based on the hinge loss, while for LR, it is derived from the log-loss function. MBGD stabilizes updates and improves convergence.
- Momentum Gradient Descent (MGD)
  Momentum-based optimization accumulates past gradients:

$$v = \beta v + (1 - \beta)\nabla_w L \quad (9)$$

$$w \leftarrow w - \alpha v \quad (10)$$

  This prevents oscillations and improves convergence speed in dynamic caching environments.
- Adam Optimizer
  Adam combines momentum and adaptive learning rates:

$$m_w = \beta_1 m_w + (1 - \beta_1)\nabla_w L \quad (11)$$

$$v_w = \beta_2 v_w + (1 - \beta_2)(\nabla_w L)^2 \quad (12)$$

$$w \leftarrow w - \alpha \frac{m_w}{\sqrt{v_w} + \epsilon} \quad (13)$$

  This technique ensures fast and stable convergence in complex caching scenarios.
- Newton's Method
  Newton's optimization inverts the Hessian matrix :

$$w \leftarrow w - H^{-1}\nabla_w L \quad (14)$$

  For LR, the Hessian is derived from the log-loss function:

$$H = \frac{1}{N}\sum_{i=1}^{N} P(y_i \mid X_i)(1 - P(y_i \mid X_i))X_i X_i^T \quad (15)$$

  This method achieves high precision but is computationally intensive. By structuring the optimization techniques in this manner, the discussion remains clear and professional, ensuring easy comparison between SVM and LR implementations.

IV. Experimental Results and Discussion

To ensure the reproducibility of our experiments and to provide a solid foundation for the comparative analysis of caching strategies in edge computing environments, this section details the dataset, preprocessing steps, data splitting methodology, model implementations, baseline algorithms, and parameter settings used throughout the study.

A. Experimental Setup

This study uses the dataset containing 5,000 real-world user requests from an operational edge computing system. Content popularity follows a Zipf distribution, reflecting real-world access patterns [12]. Each request includes response time, access duration, access frequency, file size, and data type.

Preprocessing entails: (1) Label Encoding of `data_type` utilizing `LabelEncoder`, and (2) Normalisation of numerical features employing `StandardScaler` to enhance model convergence. The data is divided using `train_test_split` in an 80:20 ratio for both Logistic Regression (LR) and Support Vector Machine (SVM) [11]. The target labels are $\{0, 1\}$ for Logistic Regression and $\{-1, 1\}$ for Support Vector Machine. Both LR and SVM are optimized with SGD, Mini-Batch Gradient Descent (MBGD), Momentum GD, Adam, and Newton's Method, in comparison to LRU and LFU baselines.

Hyperparameters: learning rate (0.1), epochs (50), batch size (32 for Mini-Batch Gradient Descent), momentum (0.9), and epsilon (1e-6 for Newton's method). Random seeds guarantee reproducibility.

TABLE I: Hyperparameters for Model Training

Parameter	Value
Learning Rate (α)	0.1
Epochs	50
Batch Size (MBGD)	32
Momentum (β)	0.9
Adam (β_1, β_2)	0.9, 0.999
Epsilon (Newton's Method)	1×10^{-6}
Random Seed	42

B. Results of Model Performance

Fig. 3 and 4 illustrate the loss curves for Logistic Regression (LR) and Support Vector Machine (SVM) models optimized using Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent (MBGD), Momentum Gradient Descent (MGD), Adam, and Newton's Method. Newton's Method attains rapid convergence, diminishing loss to nearly zero after 10–15 epochs, achieving flawless classification performance (100% accuracy, precision, and F1-Score) for both models.

In SVM **Fig. 4**, Adam and SGD show stable convergence, reaching 99.45% accuracy, 1.0 precision, and a 0.9913 F1-Score. MBGD trails slightly with 98.17% accuracy and a 0.9704 F1-Score, while Momentum GD achieves 99.08% accuracy and a 0.9854 F1-Score. For LR **Fig. 3**, SGD performs well (99.26% accuracy, 0.9884 F1-Score), while Adam shows slightly lower results (96.88% accuracy, 0.9501 F1-Score). Momentum GD underperforms (86.08% accuracy, 0.7467 F1-Score), indicating underfitting. No overfitting is observed, with performance trends matching loss reductions. These results highlight Newton's Method's superiority, while Adam and SGD offer a balance of convergence speed and model stability [15][1].

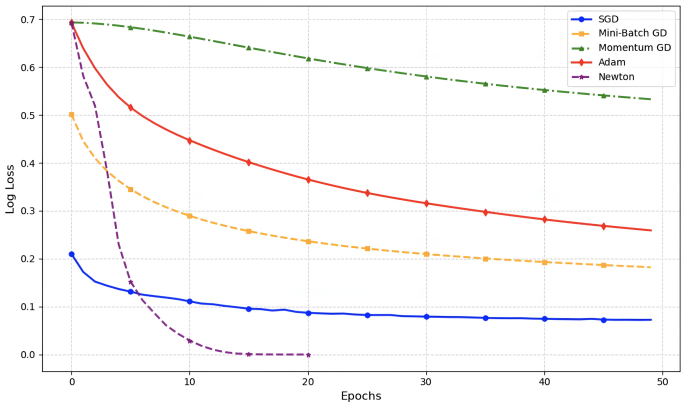

Fig. 3: Log Loss vs Epochs for Optimization Logistic Regression (LR)

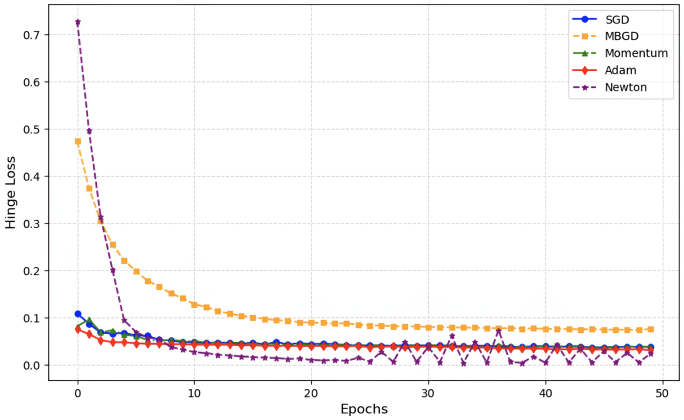

Fig. 4: Hinge Loss vs Epochs for Optimization Support Vector Machines (SVM)

C. Cache Hit Rate Analysis

Fig. 5 and 6 illustrate the cache hit rate performance of ML-based models (Logistic Regression and SVM) compared to traditional algorithms (LRU and LFU) across cache sizes of 50 MB, 100 MB, and 150 MB. The hit rate increases with larger cache sizes, but the improvement rate varies across methods. In **Fig. 5**, Logistic Regression with Momentum GD achieves the highest hit rate, around 0.13 at 150 MB, outperforming Adam and Mini-Batch GD. Similarly, **Fig. 6** shows SVM models following the same trend, with Momentum GD leading, though differences between optimizers are less pronounced. ML-based models consistently outperform LRU and LFU, which show limited adaptability despite modest improvements with larger cache sizes. The static nature of LRU and LFU restricts their

TABLE II: Comparison of LR and SVM Model Performance

Optimizer	LR (Acc/Prec/F1)	SVM (Acc/Prec/F1)
SGD	99.27 / 1.00 / 0.9884	99.45 / 1.00 / 0.9913
Mini-Batch GD	97.79 / 0.98 / 0.9677	98.17 / 1.00 / 0.9704
Momentum GD	86.08 / 0.88 / 0.7467	99.08 / 1.00 / 0.9854
Adam	96.88 / 0.97 / 0.9501	99.45 / 1.00 / 0.9913
Newton's Method	100 / 1.00 / 1.00	100 / 1.00 / 1.00

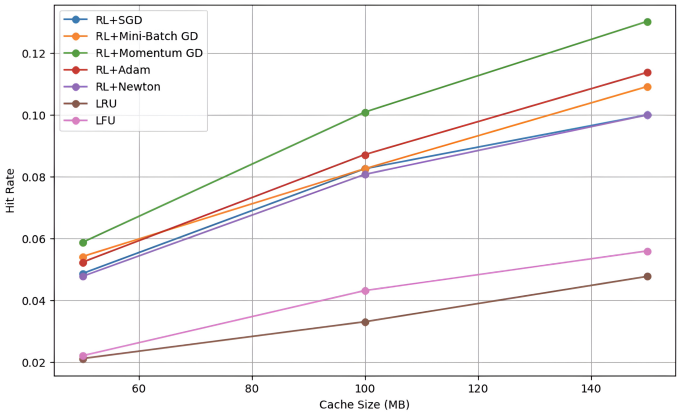

Fig. 5: Hit Rate Comparison Logistic Regresion (LR)

response to dynamic content popularity. In contrast, ML-based models optimized with Momentum GD and Adam achieve higher hit rates by adapting to real-time access patterns, enhancing cache performance in dynamic edge computing environments [8].

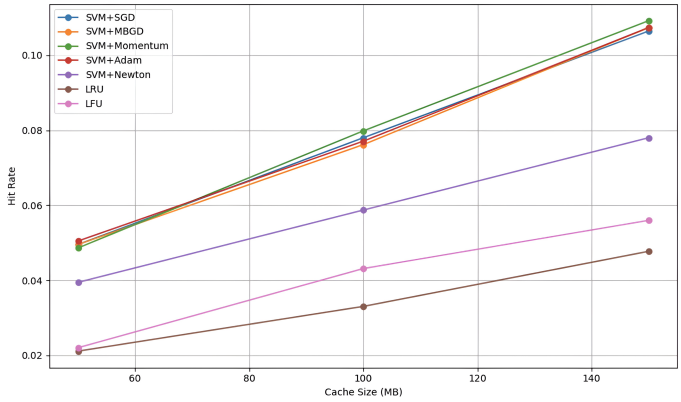

Fig. 6: Hit Rate Comparison Support Vector Machines (SVM)

D. Computational Complexity and Cost Analysis

In addition to evaluating cache placement models' performance regarding hit rate and classification accuracy, assessing their computational complexity and cost is essential. Machine learning-based caching strategies must balance predictive accuracy with computational efficiency, particularly in resource-constrained edge environments.

Support Vector Machines (SVM) exhibit higher computational complexity than Logistic Regression (LR), mainly when using kernel methods, which increase the cost to $O(n^3)$ due to matrix inversion. In contrast, LR is computationally more efficient, with a complexity of $O(ndk)$, making it more suitable for large-scale deployment. Optimization techniques further influence efficiency; SGD and Mini-Batch GD provide cost-effective training with complexities of $O(nd)$ and $O(nd/b)$, while Newton's Method, requiring Hessian inversion $(O(d^3 + nd^2))$, is computationally intensive.

Experimental results indicate that Momentum GD and Adam achieve the best trade-off between efficiency and performance, improving cache hit rates while maintaining feasible computational costs. Although Newton's Method delivers the highest classification accuracy, its price makes it impractical for real-time applications. Inference complexity remains $O(d)$ for both models, ensuring efficient decision-making in dynamic caching scenarios. Overall, selecting an optimization method should align with the computational capabilities of the edge environment, balancing predictive accuracy with real-time adaptability.

V. Conclusion

This research presents an enhanced cache placement approach for edge computing using machine learning (ML) predictive models, notably Logistic Regression (LR) and Support Vector Machines (SVM), optimized with Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent (MBGD), Momentum Gradient Descent (MGD), Adam, and Newton's Method. Experimental results demonstrate that ML-based models outperform traditional caching strategies such as Least Recently Used (LRU) and Least Frequently Used (LFU) in cache hit rate and classification accuracy [9]. Newton's Method achieves the fastest convergence, while Momentum GD delivers the highest cache hit rates, demonstrating adaptability to dynamic content demand. However, practical limitations must be considered, including the high computational complexity of Newton's Method, which may not be feasible for low-power edge devices. ML models require periodic retraining to adapt to varying network conditions, adding computational overhead. At the same time, storage and memory constraints in edge nodes pose challenges for frequent model updates and dynamic caching. Additionally, network latency and resource availability may impact real-time inference, necessitating further optimization for large-scale deployment. Future work should explore hybrid models combining ML with heuristic strategies and investigate lightweight ML models and federated learning techniques to enhance scalability and feasibility in practical edge computing applications.

References

[1] Mohsen Amidzadeh et al. "Caching in Cellular Networks Based on Multipoint Multicast Transmissions". In: *IEEE Transactions on Wireless Communications* 22.4 (2023), pp. 2393–2408. DOI: 10.1109/twc.2022.3211416.

[2] Suzhi Bi, Liang Huang, and Ying-Jun Angela Zhang. "Joint Optimization of Service Caching Placement and Computation Offloading in Mobile Edge Computing Systems". In: *IEEE Transactions on Wireless Communications* 19.7 (2020), pp. 4947–4963. DOI: 10.1109/twc.2020.2988386.

[3] Yichao Chao, Hong Ni, and Rui Han. "A Path Load-Aware Based Caching Strategy for Information-Centric Networking". In: *Electronics* 11.19 (2022), p. 3088. DOI: 10.3390/electronics11193088.

[4] Ronghui Hou et al. “Caching and Resource Allocation in Small Cell Networks”. In: *Computer Networks* 172 (2020), p. 107100. DOI: 10.1016/j.comnet.2020.107100.

[5] Sang-Eun Jeon et al. “Learning-Based Joint User Association and Cache Replacement for Cache-Enabled Cloud RAN”. In: *IEEE Open Journal of the Communications Society* 5 (2024), pp. 3038–3049. DOI: 10.1109/ojcoms.2024.3397054.

[6] Yong Ma et al. “A Fault-Tolerant Mobility-Aware Caching Method in Edge Computing”. In: *CMES-Computer Modeling in Engineering & Sciences* 140.1 (2024), pp. 907–927. DOI: 10.32604/cmes.2024.048759.

[7] Sajad Mehrizi et al. “Online Spatiotemporal Popularity Learning via Variational Bayes for Cooperative Caching”. In: *IEEE Transactions on Communications* 68.11 (2020), pp. 7068–7082. DOI: 10.1109/tcomm.2020.3015478.

[8] Lubna Mohammed et al. “Performance of Cache Placement Using Supervised Learning Techniques in Mobile Edge Networks”. In: *IET Networks* 10.6 (2021), pp. 304–321. DOI: 10.1049/ntw2.12029.

[9] Nazib Abdun Nasir and Seong-Ho Jeong. “Content Management Based on Content Popularity Ranking in Information-Centric Networks”. In: *Applied Sciences* 11.13 (2021), p. 6088.

[10] Milena Radenkovic and Vu San Ha Huynh. “Cognitive Caching at the Edges for Mobile Social Community Networks: A Multi-Agent Deep Reinforcement Learning Approach”. In: *IEEE access* 8 (2020), pp. 179561–179574. DOI: 10.1109/access.2020.3027707.

[11] Bhisham Sharma et al. “Emerging Sensor Communication Network-Based AI/ML Driven Intelligent IoT”. In: *Sensors* 23.18 (2023), p. 7814.

[12] Muhammad Sheraz et al. “A Reinforcement Learning Based Data Caching in Wireless Networks”. In: *Applied Sciences* 12.11 (2022), p. 5692. DOI: 10.3390/app12115692.

[13] Muhammad Sheraz et al. “Mobility-Aware Data Caching to Improve D2D Communications in Heterogeneous Networks”. In: *Electronics* 11.21 (2022), p. 3434. DOI: 10.3390/electronics11213434.

[14] Shimin Sun et al. “An on-Demand Collaborative Edge Caching Strategy for Edge–fog–cloud Environment”. In: *Computer Communications* 228 (2024), p. 107967. DOI: 10.1016/j.comcom.2024.107967.

[15] Wuqu Wang et al. “Fundamental Limits of Coded Caching in Request-Robust D2D Communication Networks”. In: *Entropy* 26.3 (2024), p. 250. DOI: 10.3390/e26030250.

[16] Ran Zhang et al. “Deep Reinforcement Learning (DRL)-Based Device-to-Device (D2D) Caching With Blockchain and Mobile Edge Computing”. In: *IEEE Transactions on Wireless Communications* 19.10 (2020), pp. 6469–6485. DOI: 10.1109/twc.2020.3003454.

[17] Tiankui Zhang et al. “Cache-Enabling UAV Communications: Network Deployment and Resource Allocation”. In: *IEEE Transactions on Wireless Communications* 19.11 (2020), pp. 7470–7483. DOI: 10.1109/twc.2020.3011881.

[18] Mulki Indana Zulfa et al. “Accelerating Convergence in Data Offloading Solutions: A Greedy-Assisted Genetic Algorithm Approach”. In: *International Journal of Robotics and Control Systems* 4.4 (2024), pp. 1934–1946. DOI: 10.31763/ijrcs.v4i4.1652.