National College of Ireland

Artificial intelligence-based Cache Partitioning for protecting the systems against vulnerabilities

MSc Research Project
MSc Cybersecurity

Angel Maroor Jose
Student ID:22213201

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Angel Maroor Jose
Student ID:	22213201
Programme:	MSc Cybersecurity **Year:** 2023-2024
Module:	MSc Research Practicum
Supervisor:	Niall Heffernan
Submission Due Date:	13/09/2024
Project Title:	Artificial intelligence-based Cache partitioning to protect the system against Vulnerabilities
Word Count:	7110 **Page Count:**25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Angel Maroor Jose

Date: 12/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Artificial intelligence-based Cache Partitioning for protecting the systems against vulnerabilities

Angel Maroor Jose
22213201

Abstract

Efficient cache management is vital in contemporary computer systems to optimize performance by minimizing latency and maximizing efficiency. Cache partitioning is a method that distributes cache resources across many processes or threads to guarantee equitable and effective consumption while reducing cache congestion. The goal of this project is to create a new cache partitioning method that utilizes artificial intelligence (AI) to improve cache use. This will result in higher hit rates and lower access latency. The suggested approach observes cache behavior and implements partitioning exclusively when abnormal or significant cache patterns are identified, therefore reducing RAM usage and releasing capacity. The system selectively intervenes by utilizing machine learning models to identify detrimental cache behaviors, resulting in improved speed and security. Out of the different models assessed, XGboost, Random Forest and DNN had the greatest performance, obtaining a 93% accuracy across all evaluation criteria. The paper further examines existing research on cache partitioning, vulnerability identification, and the use of artificial intelligence in the field of cybersecurity. The results emphasize the capability of this method to enhance the efficiency and safety of a system in intricate, multi-threaded settings.
Keywords: Cache Partitioning, Machine Learning based systems, Suspicious Detection, RAM storage

1: Introduction

In modern computing systems, caching plays a critical role in enhancing performance by keeping commonly accessed data closer to the cpu, thereby reducing latency and improving efficiency. However, as applications and workloads become more complex and diverse, managing cache resources effectively becomes a significant challenge. This is where cache partitioning comes into play (**Qiu, J., Hua, Z., Liu, L., Cao, M. and Chen, D., 2022**).

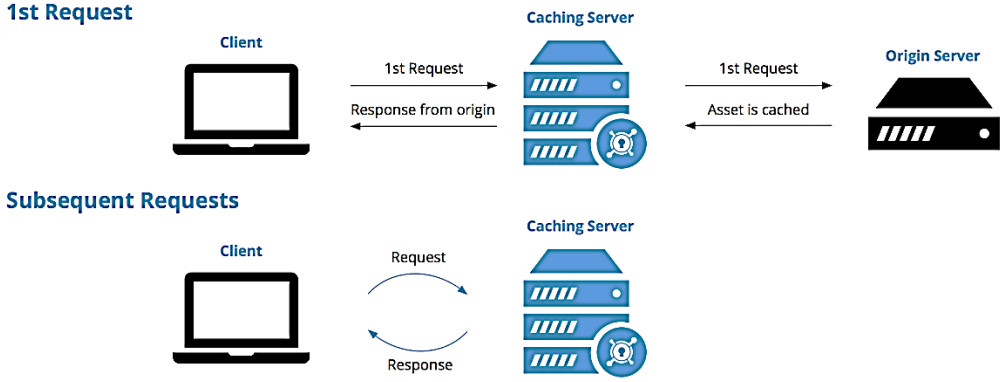


Figure 1: Cache Definition and Explanation (**Source: KeyCDN**)

1.1 Motivation

Cache partitioning as illustrated in Fig 1, is a strategic method used to distribute cache resources among different processes to guarantee fair and efficient utilization. Dividing the cache into several segments or partitions will reduce the cache contention and interference. This approach will ensure that all applications or processes receive a particular amount of cache, this method will improve the overall system functionality and reliability and prevent situations where the needs of one workload adversely affect those of other workloads. (**Wang, Z. and O'Boyle, M.F., 2010**).

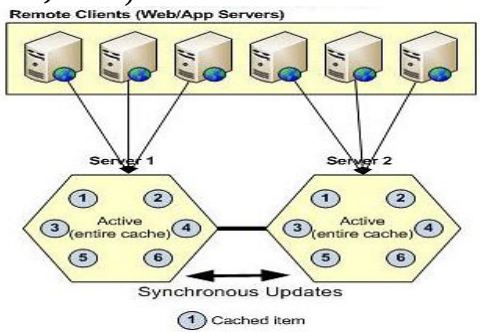


Figure 2: An example of replicated Cache (**Source: Alachisoft**)

The concept of cache partitioning is extremely important in occasions involving several multi-core processors and multi-threaded environments, as shared cache can result in an overload. Effective cache usage, fewer cache misses, and improved cache performance are all made feasible by optimal cache partitioning also encourages cache performance predictability (Alsaade, F.W. and Al-Adhaileh, M.H., 2023). Furthermore, it can assist in fulfilling quality of service requirements for a wide range of applications running simultaneously on the same machine.

1.2 Aim of the Research

The goal of this project is to create a cache partitioning scheme using machine learning. This scheme will optimize cache usage by dynamically categorizing and handling data requests. The goal is to enhance system performance by increasing cache hit rates and decreasing access latencies. In this the system that is proposed uses the machine learning models to learn the harmful cache and, in such scenarios, deploy the machine learning aided-cache partitioning which partitions the cache and helps in mitigating the vulnerability (**Liu, W., Cui, J., Li, T., Liu, J. and Yang, L.T., 2022**).

1.3 Research Objective

This introduction to cache partitioning will explore the principles behind cache partitioning, its benefits, and the various techniques used to implement it. By understanding and applying cache partitioning strategies, system designers and engineers can optimize cache performance and enhance the overall efficiency of computing systems. Recent trends towards the development of machine learning-aided cache partitioning have largely influenced cyber security. A caching strategy (**Sun, H., Sun, C., Tong, H., Yue, Y. and Qin, X., 2024**) is implemented using a random forest classifier to classify I/O requests into three categories: critical, intermediate, and non-critical. Cache, when implemented in an SSD emulator, surpasses other caching schemes such as LRU, CFLRU, LCR, NCache, ML_WP, and Cache_ANN in terms of reaction time, write amplification, erase count, and hit ratio. Its performance is just slightly different from the OPT strategy.

1.4 Research Questions

RQ: How can a machine learning-based cache partitioning technique enhance cache hit rates and system performance in dynamic and diversified workload conditions compared to standard caching solutions?

Solution: The concept for this is to develop the complex system that involves multiple machine learning models, cache partitioning techniques, and training processes into a vulnerability-based deployment or cache partitioning so that the computation, as well as the storage space, is minimized. Hence in this, are not only getting the suspicious rates to decrease but also optimizing the solution.

1.5 Thesis Structure

The following chapter will discuss the different research done towards AI advancement in the detection of malicious cache in a system. The methodology chapter will discuss different machine learning and cache partitioning methods. The chapter on implementation will discuss the proposed system setup and the process of implementation as the name suggests. The chapter on results and analysis will discuss and analyze the results that are found.

2. Literature Review

2.1 Advancements in cache partitioning

(**Wang et al. 2024**) developed an ε-LAP, an innovative cache partitioning technique for Content Delivery Networks(CDNs). This is developed to maximize the cache performance, including complex workloads and traffic. Based on average hit numbers ε-LAP uses shadow caches, for each partition. This enables effective storage capacity transfers between partitions when scaling is needed. Without compromising the performance this threshold parameter, ε, minimizes the unnecessary partition by resizing into 96.8%.

(**Park et al., 2020**) To address the issues of cache pollution in shared last-level caches(LLC) in multi-core systems, propose a cache partitioning methodology termed as page reusability-based cache partitioning(PRCP). By reducing the cache contamination through dynamic cache partitioning based on page reusability, PRCP maximizes cache efficiency. Highly reused and lowly-reused pages the authors will categorize pages based on how frequently they are scanned via page table. By using the page coloring technique for allocating different regions for unique cache the PRCP algorithm prevents data with weak temporal locality and strong temporal locality.

This research suggested that (**Wang et al., 2023**) developed KRR, a probabilistic stack approach that precisely mimics the random sampling-based LRU methodology used in in-memory key-value caches such as Redis. Knowledge representation and Reasoning may accept the objects of both fixed and variable sizes. Moreover, to reduce the time in run introduced a highly effective method for updating the stack. kRedis was introduced as a memory partitioning strategy. This will automatically adjust the memory sampling size based on reference locality and latency. The evaluation results showed the fact that it reduces average access latency by 50.2% and increases throughput by 262.8%.

(**Holtryd et al., 2023**) present SCALE, to prevent timing side channel-based attacks in dynamically partitioned last-level caches, which is a cache allocation approach. SCALE includes random components into cache allocation decisions by creating noise, preventing attackers from recognizing predictable placement changes, and extracting critical

information. The approach (Holtryd et al., 2023) employs differential privacy to provide quantitative and information-theoretic guarantees of security. Their evaluations show that SCALE not simply improves confidentiality but also outperforms current secured cache methods in terms of speed on a 16-core tiled chip multi-processor with multi-programmed programs.

2.2 AI-based vulnerability detection

(**Lin et al., 2023**) provide VulEye, a new vulnerability detection method for PHP applications that use a Graph Neural Network. VulEye generates the Programme Dependence Graph (PDG) for the PHP source code, partitions the PDG into sub-graphs called Sub-Dependence Graphs (SDGs) by isolating critical functions, then utilises these SDGs as inputs to train a Graph Neural Network model. The model consists of three stack units: a Graph Convolutional Network (GCN) layer, a Top-k pooling layer, and an attention layer. These are followed by a Multilayer Perceptron (MLP) and a softmax classifier, which are used to forecast whether the Sustainable Development Goal (SDG) is susceptible. The assessment of the PHP vulnerability test suite in the Software Assurance Reference Dataset reveals that VulEye attains a macro-average F1 score of 99% in binary classification and 95% in multi-class classification.

In this study, (**Purba et al., 2023**) run a series of tests to assess the effectiveness of four renowned Large Language Models (LLMs) in identifying software vulnerabilities. They utilize two widely recognized public datasets for this purpose. Their findings demonstrate a notable disparity in performance between these LLMs and widely used static analysis techniques, principally attributable to the LLMs' elevated rates of false positives. LLMs exhibit significant potential in detecting minor patterns frequently linked to software vulnerabilities. This indicates a favorable direction for progress by combining LLMs with other program analysis approaches to improve the detection of software vulnerabilities.

(**Alzahrani and Alenazi, 2023**) introduce an advanced, state-of-the-art intrusion detection system (IDS) that operates in real-time and is specifically tailored for software-defined networking (SDN). Two datasets were generated utilizing Mininet and the Ryu controller. These datasets included typical traffic as well as other forms of attacks, such as Fin flood, UDP flood, ICMP flood, OS probe scan, port probe scan, TCP bandwidth flood, and TCP syn flood. The datasets were utilized for training various supervised binary classification machine learning algorithms, such as the k-nearest neighbor, AdaBoost, decision tree (DT), random forest, naive Bayes, multilayer perceptron, support vector machine, and XGBoost.

2.3 AI and Software techniques in advancements of Hacking detection and Cache Partitioning

In research, **Zhang et al. (2024)** study the usefulness of different prompt designs in using the ChatGPT to identify software vulnerabilities. The authors utilized the capabilities of ChatGPT as well as improved studies by increasing fundamental impulses and adding organizational and logical data in addition to them. The effectiveness of these prompt-enhanced techniques in Comprehensive testing verifies the vulnerability detection performance of ChatGPT conducted on two vulnerability datasets.

Dhanya et al. (2023)) Investigate the effectiveness of various strategies for detecting network attacks that employ machine learning and deep learning models. They particularly concentrate on the UNSW-NB15 dataset. The authors evaluate traditional machine learning approaches such as Decision Trees, Random Forest, AdaBoost, and XGBoost, as well as a K-Nearest Neighbour classifier and a deep learning model composed of two dense layers with

ReLU activation and a third layer with Sigmoid activation. The models are trained and tested using a dataset that includes 49 distinct attributes from nine different types of attacks. The major goal is to obtain high levels of accuracy in identifying network breaches.

Wang et al. (2023) In this detailed research, they look at the privacy and security challenges associated with mobile-edge computing (MEC) from an AI perspective. In this study, they create a robust foundation for MEC service platforms by merging SDN and NFV, as seen in the ETSI MEC standard design. In this post, we'll explore at some of the new privacy and security concerns surrounding MEC, as well as how artificial intelligence (AI) solutions such as adaptive security measures and anomaly detection can help. Their findings pave the way for further studies that could employ AI to improve MEC deployment security and reduce dangers, which would be fantastic for the Internet of Things and other low-latency applications**.**

Stutz et al. (2024) look into the complexities of cyber-physical systems (CPS) and the evolving risks posed by cyber threats. The authors notice that CPS are becoming more complex and self-sufficient as a result of many security weaknesses such as information channels, hardware fraud, and virtual machine intrusion. To overcome these flaws, the authors suggest a CPS-AI system that employs AI to improve CT models and security measures. The paper also discusses the limitations of traditional intrusion prevention security systems (IPSSs) in dealing with CPS vulnerabilities and proposes advanced techniques for detecting and mitigating CTs, such as nonlinear surveillance systems based on neural networks (NNs) and variable structure control (VSC).

This study underlines the need for Both the offensive and defensive uses of machine learning in cybersecurity explored by **Shang (2024)**. The research delves into the ways in which machine learning enables cyberattacks, such as the creation of smart botnets, spear phishing, and the introduction of covert malware. The article goes on to say that machine learning is useful for cyber defense, especially when it comes to finding and stopping these cyber dangers. By analyzing malware, assessing network vulnerabilities, and predicting future threats, it highlights the importance of artificial intelligence in improving digital safety. Additionally, in light of the difficulties brought about by the ever-changing nature of cybercrime in this age of fast digital transformation, the study investigates potential countermeasures to cyber threats that target machine learning models.

In this 2023 study, (Ding et al., 2023) developed a new defense strategy dubbed the Machine-learning-based Attack Detection Scheme (MADS) to safeguard Continuous-Variable Quantum Key Distribution (CVQKD) systems from quantum hacking attempts. The method uses Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to remove noise and identify outliers in attack-related feature vectors. Following that, Multiclass Support Vector Machines (MCSVMs) are used to categorize and predict the processed data, allowing for quick decisions on the creation of final secret keys. The simulation results show that MADS is successful in detecting and lowering quantum hacking risks, therefore boosting the security of CVQKD systems by correcting inflated secret key rates without relying on pre-established defense strategies.

2.4 Cloud and Network Framework

(**Qiu et al., 2022**) offer a method called Classification-and allocation for dividing the last level cache in the environments. This method tries to overcome conflicts caused by distributing resources by offering a lightweight solution. The technique uses a support vector

machine(SVM) to divide programs into three different classes based on their performance change characteristics(PCC). Moreover, it deploys a Bayesian optimizer to effectively contribute LLC resources. C&A optimizes cache consumption by making sure that programs with similar PCC use the same section of the LLC. This method is used for testing various workloads and outperforms the state-of-the-art Kpart method in terms of total system performance and fairness by 7.45% and 22.50 % respectively. Additionally, it reduce allocation inefficiencies by 20.60%.

For OpenStack cloud infrastructures (**Krishnan, P., Jain, K., Aldweesh, A., Prabu, P. and Buyya, R., 2023**) that use software-defined networking (SDN), the authors present OpenStackDP, an all-inclusive security framework. To improve network management and security, this framework uses a mix of SDN, NFV, and ML/AI techniques. Integrating intelligent anomaly detection sensors and lightweight monitoring capabilities into the data plane is at the heart of OpenStackDP. Virtual machines and cloud-based applications are constantly watched by these sensors. An analytics engine that uses machine learning and artificial intelligence to spot abnormalities or possible dangers runs on network co-processors or hardware in the switches and examines the streaming data in real-time. Quick decisions and defensive measures, like deploying virtual network functions (VNFs), can be taken in response to risks based on this analysis, preventing them from affecting tenant compute nodes or long-term data stores. By achieving rapid threat detection and response, the framework hopes to greatly enhance security posture, leading to better QoS and quicker recovery from cyber-attacks.

An edge-cloud collaboration-based communication attack detection framework for Cyber-Physical Systems (CPS) is proposed by **Chen, C., Li, Y., Wang, Q., Yang, X., Wang, X., and Yang, L.T. , 2023** to address the growing communication security concerns in CPS networks. To better detect security breaches and manage the high computational complexity and vast data created by IoT devices, this framework uses deep learning technologies in conjunction with cloud and edge computing. The goal is to improve hardware resource parallelism and satisfy the processing needs of large-scale hierarchical CPS attack detection in real-time. Experiments in simulation conducted by the authors proved that the suggested framework worked as intended, proving that it might increase cloud collaboration and the intelligence of physical devices. Both the safety and efficiency of CPS communication networks are enhanced by this method.

The authors of the paper suggest a strongly Boosted Neural Network as a means of detecting cyberattacks that involve multiple stages (**Dalal, Manoharan, Lilhore, Seth, Mohammed Alsekait, Simaiya, Hamdi, and Raahemifar, K., 2023**). Finding novel, complicated threats, such as zero-day attacks and multi-step assaults, is a problem that their method attempts to solve. After testing their Extremely Boosted Neural Network against other machine learning methods, the authors concluded that it performed the best. The Quest Model achieved a prediction accuracy of 94.09% for multi-stage cyber-attacks, the Bayesian Network 97.29%, and the Neural Network 99.09%. With the help of the Multi-Step Cyber-Attack Dataset (MSCAD), the suggested model achieved a remarkable 99.72% accuracy rate. These findings demonstrate that the Extremely Boosted Neural Network has great promise for handling cyber assaults in communication settings where they occur in real-time.
(**Zhao et al., 2023**) propose a Content-Adaptive Cache Partitioning (CACP) technique for mobile edge networks. The objective and motto is to reduce the cost of obtaining different content as due to the growing popularity of mobile video services. The authors propose a two-tier caching scheme aiming at addressing cache redundancy and user mobility; this

assumes caching at the 5G base station gNB and caching at device to device (D2D caching). gNBs utilize a public and private cache partitioning mechanism to improve the availability of local cache. Also, different kinds of technological devices such as mobile devices have a custom cache partitioning technique that is both static and dynamic and can be programmed to fit various mobility patterns. However, simulations show that CACP always offers low costs for content access and a high number of successful hits; unfortunately, it does result in rather high power consumption by its users.

2.5 Summary

The literature study examines current progress in cache partitioning, AI-driven vulnerability identification, and cybersecurity. Significant research includes ε-LAP for the dynamic resizing of caches, PRCP for the reduction of cache pollution, and KRR for enhancing the efficiency of key-value caches. The incorporation of machine learning in security is exemplified by AI approaches such as VulEye for PHP vulnerability identification, as well as improved intrusion detection systems for SDN and CPS. The essay also goes over novel approaches like SCALE for reducing side-channel attacks, AI-enhanced MEC privacy solutions, and hybrid models for identifying cyber-attacks. These findings emphasize the critical role of AI in increasing cache performance and strengthening cybersecurity.

2.6 Focused Research Domain

The study presents an innovative AI-driven cache partitioning method that increases cache utilization by interfering only when suspicious or significant cache patterns are identified. Unlike traditional systems that function continuously this approach decreases RAM utilization and frees storage, hence enhancing the efficiency of the system. The system employs machine learning models to identify harmful cache behaviors and strategically uses cache partitioning to enhance both performance and security. This method not only boosts cache hit rates but also successfully addresses cybersecurity challenges, making it a unique addition to the discipline of cache management and AI-driven cybersecurity.

3. Research Methodology

During the model creation phase, assess many machine learning classifiers, such as Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines. When employing deep learning methods, utilize neural networks such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) (**Grossberg, S., 2013**) to effectively capture intricate patterns within the data. The chosen models are further trained using the pre-processed dataset, employing cross-validation techniques to ensure reliable performance across various data subsets. Hyperparameter tuning is carried out to enhance the efficiency of a model by utilizing Grid Search or Random Search to identify the optimal configuration.

3.1 Dataset Overview

Web applications attack datasets for machine learning are somewhat scarce. Despite the expectation of abundant collections of parsed legitimate and anomalous traffic from http logs, such datasets are not readily available (**Vartouni, A.M., Kashi, S.S. and Teshnehlab, M., 2018**). It is believed that the majority of individuals and organizations who gather this information choose to keep it confidential to use it for their search engines and profit from selling a product.

```
POST http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=AE29AEEBDE479D5E1A18B4108C8E3CE0
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 146

id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHER
```

Figure 3: An example of the tuple from the original dataset for CSIC anomalous traffic (Source: CSIC)

This dataset consists of 60,000+ records and consists of features such as Method, User-Agent, Pragma, Cache-Control, Accept, Accept-encoding, Accept-charset language, host, cookie, content-type, connection, length, content, classification, and URL. A benefit of this data is the inclusion of both GET requests with query data and POST requests with request data. Nevertheless, there are some drawbacks associated with this data. The CSIC dataset is only derived from a controlled laboratory setting, resulting in identical HOST information across all instances.

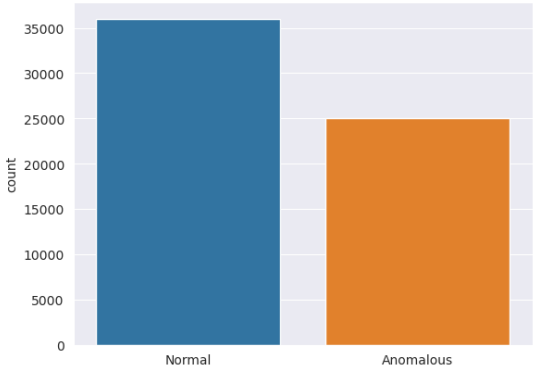


Figure 4: The counts of the different values in the output feature.

The above figure shows that there are approximately 35000 normal cache information and 25000 anomalous caches.

3.2 Data Exploration and Pre-Processing

Data exploration is the preliminary stage of data analysis, involving a thorough examination of a dataset to get an understanding of its contents. It is like conducting detective work on data, revealing its attributes, trends, and potential issues. Data exploration is an essential component of data analysis as it enables the discovery of valuable insights that are concealed within the data. Most common URLs:

1. URL: http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1 - Count: 2441
2. URL: http://localhost:8080/tienda1/publico/autenticar.jsp HTTP/1.1 - Count: 2422
3. URL: http://localhost:8080/tienda1/publico/registro.jsp HTTP/1.1 - Count: 2417
4. URL: http://localhost:8080/tienda1/miembros/editar.jsp HTTP/1.1 - Count: 2412
5. URL: http://localhost:8080/tienda1/publico/pagar.jsp HTTP/1.1 - Count: 2379
6. URL: http://localhost:8080/tienda1/publico/caracteristicas.jsp HTTP/1.1 - Count: 2003
7. URL: http://localhost:8080/tienda1/publico/vaciar.jsp HTTP/1.1 - Count: 1965
8. URL: http://localhost:8080/tienda1/publico/entrar.jsp HTTP/1.1 - Count: 1938

9. URL: http://localhost:8080/tienda1/index.jsp HTTP/1.1 - Count: 1000
10. URL: http://localhost:8080/tienda1/miembros/salir.jsp HTTP/1.1 - Count: 1000

Patterns & Trends: Do repeating themes or links exist across various data points?

Anomalies: Are there any data points that deviate from the anticipated range, maybe suggesting mistakes or outliers?

Analyzing URLs includes the following things,

1. Count characters: Check the unusual number of dots, slashes, or hyphens in URLs to spot anything suspicious.
2. Examine structure: Based on how many folders or embedded links a URL has, and if parts are encoded (like 20% for spaces) which might hide URLs' true purpose.
3. Measure length: Look at how long and how many numbers the URL has.
4. Spot Risky Patterns: Search for certain words that might show potential fake or harmful websites, and check if the URL is shortened, which could hide its real destination.

After this, the cookies are removed from each of the samples such that the feature does not discriminate or correlate.

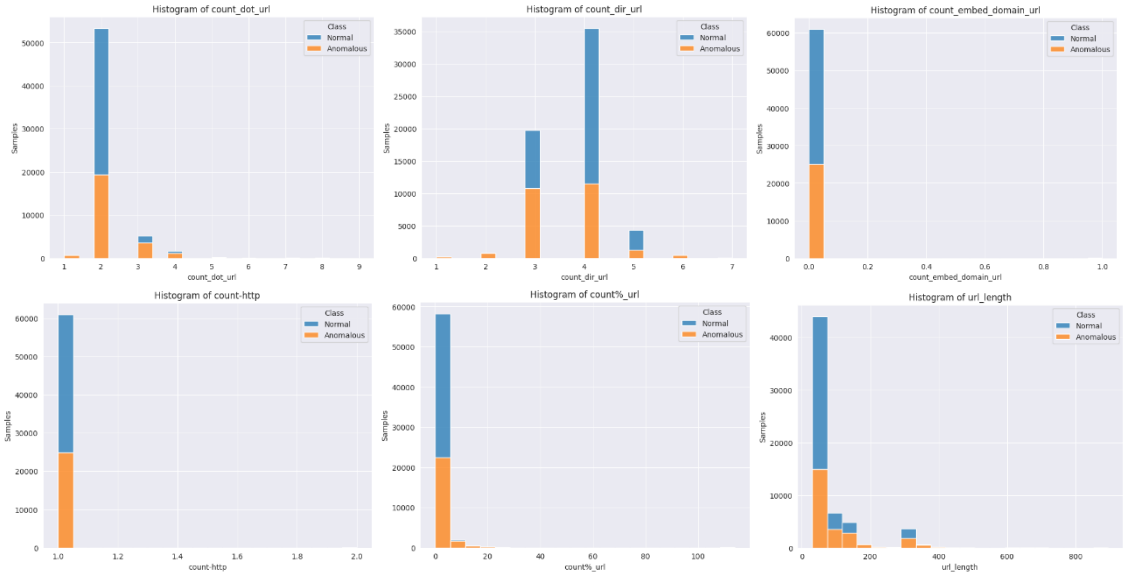


Figure 5: Histogram of different features

The above figures are trying to understand the distribution of two class values, Normal and

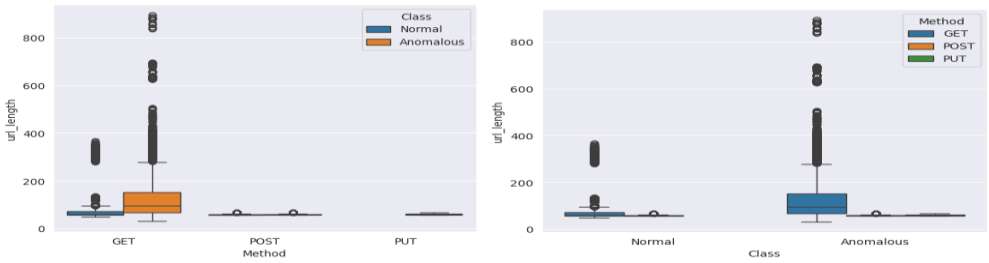


Figure 6: Outliers in the two methods

In the above figure, the points that are outside the box are the outliers which can either be removed or corrected.

3.3 Modelling

In the model development phase, evaluate various machine learning classifiers, including Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines. For deep learning approaches, consider neural networks like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to capture complex patterns in the data. The selected models are then trained using the pre-processed dataset, with cross-validation techniques employed to ensure robust performance across different data subsets. To optimize model performance, hyperparameters are tuned, and the optimal configuration is found using Grid search or Random search. Model evaluation is the process of examining the performance of trained models using metrics such as Accuracy, precision, Recall, F1 score, and the receiver Operating Characteristics(ROC) curve. These parameters provide information about the model's ability to appropriately categorise normal and malicious requests. Validation is performed on a separate test set to assess how well the model works in real-world circumstances. Additional testing verifies that the model is robust against a variety of web threats.

3.3.1 Sampling

Hold-out (**Schorfheide, F. and Wolpin, K.I., 2012**) refers to the process of dividing a dataset into two separate sets, namely the 'train' set and the 'test' set. The training set is the data on which the model is trained, whereas the test set is employed to evaluate the model's performance on new and unknown data. When employing the hold-out approach, it is customary to allocate 80% of the data for training purposes and save the remaining 20% for testing.

Cross-validation (Berrar, D., 2019), often known as 'k-fold cross-validation', involves randomly dividing the dataset into 'k' groups. One group is designated as the test set, while the remaining groups are utilized as the training set. The model is trained using the training set and evaluated using the test set. Then the process is continued until each distinct group has been utilized as the test set. For instance, in the case of 5-fold cross-validation, the dataset is divided into 5 distinct groups. The model is then trained and tested 5 times independently, ensuring that each group has an opportunity to serve as the test set.

3.3.2 Random Forest

The Random Forest algorithm (**Rigatti, S.J., 2017**) is a reliable method for tree-based learning in the area of Machine Learning. The system develops a large number of Decision Trees in the training phase. Every tree is built by selecting from a portion of the dataset to evaluate a random selection of characteristics in each division. The application of randomization in this scenario generates variation among each tree, limiting the danger of excessive fitting and improving the overall accuracy of predictions.

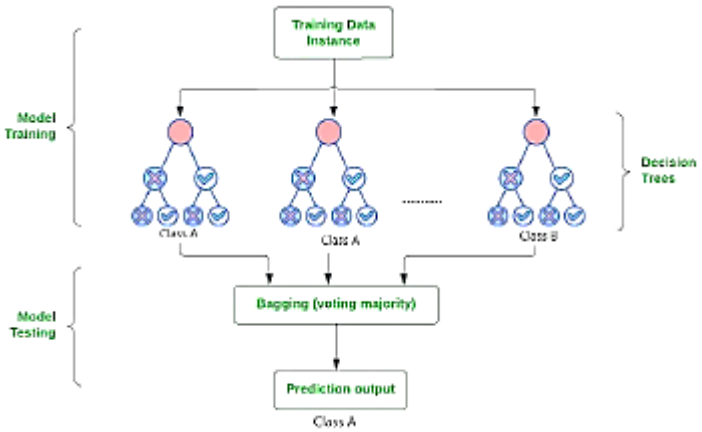


Figure 5: Random Forest flow diagram (**Source: GeeksforGeeks**)

3.3.3 K-Nearest Neighbors

KNN (**Peterson, L.E., 2009**) is an important classification approach in the field of machine learning. It belongs to the category of supervised learning and is commonly used in pattern recognition, data mining, and intrusion detection. It is particularly useful in real-world scenarios since it is random, which means it does not make any presumptions about the distribution of data. Given a set of previous data, commonly referred to as training data, we have coordinates that are sorted into groups based on a given property.

3.3.4 Decision Tree

Decision Trees (DTs) (**De Ville, B., 2013**) are an irregular supervised learning approach that might be employed for both regression and classification. The goal is to build a model that can predict the value of a target variable using fundamental decisions derived from the data's properties. A tree can be thought of as a representation that approximates a function by splitting the space of inputs into areas and allocating values that remain to each.

3.3.5 Gradient boosting (**Chen, T. and Guestrin, C., 2016**) is a machine learning ensemble technique that methodically integrates the predicted outcomes of multiple weak learners, typically decision trees. The goal is to improve overall forecast performance by optimising model weights based on errors from previous iterations. This technique steadily minimizes mistakes in prediction and increases the model's performance.

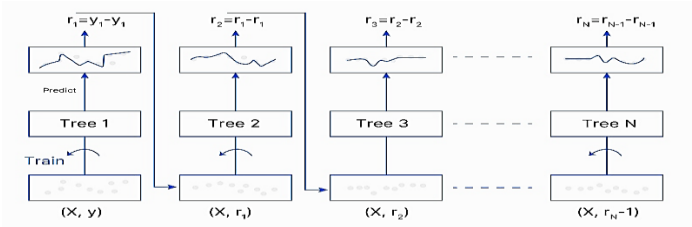


Figure 7: The example showing how the errors are propagated in the structure of the serial tree and how the weights are formularized to minimize them towards getting the actual results (**Source: TowardsDatascience**)

3.3.7 Support Vector Machines

A support vector machine (SVM) (**Wang, H. and Hu, D., 2005**) is a machine learning approach that solves challenging problems in outlier identification, regression, and classification using supervised learning models. It does this by employing predefined classes,

labels, or outputs to carry out efficient data transformations that draw boundaries between data points. The SVM approach's main goal is to locate a hyperplane that successfully divides data points from various classes.

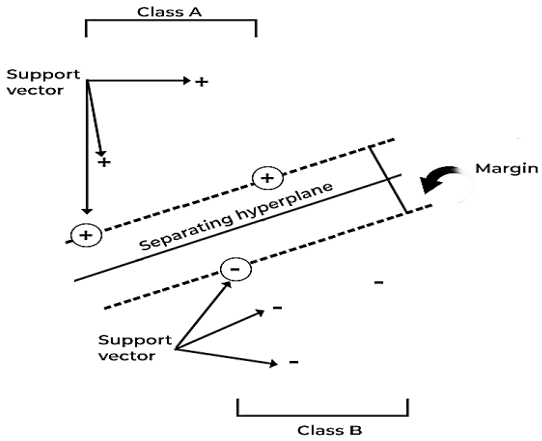


Figure 9: An example of maximizing the gain margin using the support vectors in SVM (**Source: Analytics Vidhya**)

3.3.8 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) (**Ketkar, N. and Ketkar, N., 2017**) is a modified version of the Gradient Descent method that was designed particularly to improve machine learning models. It overcomes classic Gradient Descent methods' processing inefficiencies when working with huge datasets in machine learning applications. Stochastic Gradient Descent (SGD) use a single randomised training sample or a few samples for each iteration, rather than a complete data set.

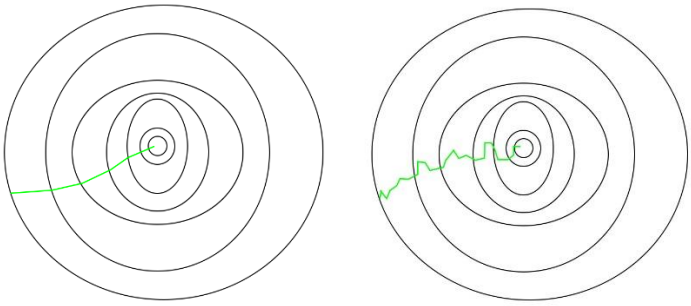


Figure 9: (a) The path taken by Batch Gradient Descent (b) A path taken by Stochastic Gradient Descent (**Source: Geeks for Geeks**).

3.3.9 Deep Neural Network

A deep neural network (DNN) (**Kelleher, J.D., 2019**) is an artificial neural network that has numerous layers between the input and output. Neural networks are classified into various categories, although they all share fundamental characteristics such as neurones, synapses, weights, biases, and functions. These components combine to simulate human brain activity and can be trained in the same way as any other machine learning method.

3.3.10 Cache Partitioning

Cache partitioning is a method used to enhance the predictability of the behavior of the instruction cache (I-cache) (**Qiu, J., Hua, Z., Liu, L., Cao, M. and Chen, D., 2022**). Every job inside a system is allocated to a distinct cache partition. Tasks in this system can only

remove cache lines that are located in the partition they are assigned to. Therefore, the cache is no longer affected by context shifts, allowing numerous processes to run simultaneously without interference. This enables the use of static Worst-Case Execution Time (WCET) assessments for each job in the system in isolation. The total Worst-Case Execution Time (WCET) of a multi-task system that uses partitioned caches is determined by adding the WCETs of each job with a certain partition size, together with the additional time necessary for scheduling, including the overhead for context changes.

3.3.11 Evaluation Matrix

Metric	Formula
True positive rate, recall	$\frac{TP}{TP+FN}$
False positive rate	$\frac{FP}{FP+TN}$
Precision	$\frac{TP}{TP+FP}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
F-measure	$\frac{2 \cdot precision \cdot recall}{precision + recall}$

3.3.12 Deployment Method

In this project, will try to formulate a streamlit application in which the solution is deployed locally. When the streamlit application is run using the command,
Streamlit run <<code_name.py>>, a local host is opened showcasing the working of the solution. The response system (A fully deployable code using the trained machine learning model) helps in understanding the suspicions and then deciding on when to call the block containing the cache partitioning. This solution is an integration of artificial intelligence and cache partitioning to optimize the computation as well as the service requirements.

4: Design Specification

This project plans to make a machine learning-infused solution for cache partitioning to mitigate the vulnerabilities that arise due to the cache. A cache for example is as below,

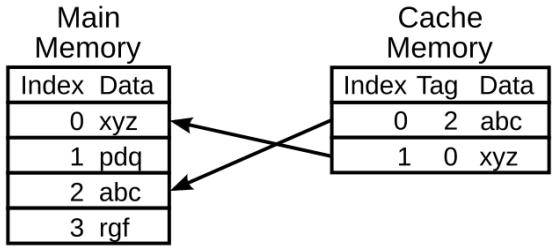


Figure 11: An example of the cache, the cache stores the data temporarily
To mitigate any vulnerabilities that might arise due to cache, we introduce two important steps,

a. Stage 1: Understand the probability of vulnerability arising due to cache by training a machine learning model and then deploying a trained model locally
b. Stage 2: Make a cache partitioning using python and call it when there is a suspicion seen.

Once stage 1 is trained and deployed, it is executed along with the cache partition. In this, a streamlit-based response system is created and deployed. The machine learning model that can be deployed can be lightweight like boosting models. Once the app is executed it can be viewed in your browser, which will open automatically.

Local URL: http://localhost:8501

Network URL: http://192.168.0.147:85

5. Implementation

5.1 Machine Learning Framework for identifying the suspicious cache

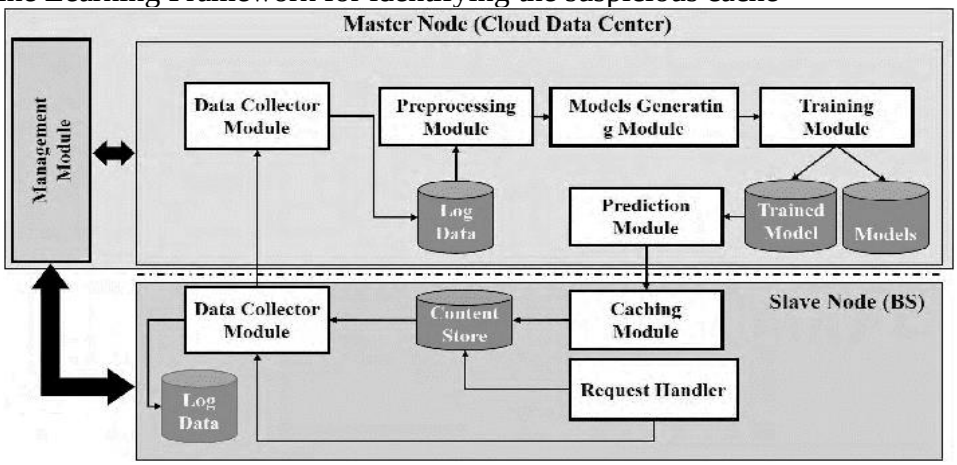


Figure 12: An intelligent Cache partitioning framework with the use of machine learning trained on the cache history

Algorithm Flow

Step 1: Data Collection – After thorough research, it was found that the CSIC dataset for this research. In real-time, the locally stored cache is the data that will be fed into the software.

a. Extraction of the features from the cache
b. Imputing if required

Step 2: Data Pre-Processing – Identifying the important pre-processing steps for the points where there are outliers or missing values. The common imputation techniques used are as below,

a. Outliers: Using the Box Plot, the extreme values are imputed from the datasets
b. Missing Values: The missing values are imputed with,
    a. Continuous Values: Imputed with the mean or median
    b. Categorical Values: Imputed with mode

Step 3: Data Sampling – In this research, a strategy for using both the holdout as well as the k-fold cross-validation techniques. In the case of training the model, we have used the holdout method with 80% training and 20% testing. While fixing the hyper-parameters, using k=5 folds for a better understanding of the hyperparameters

Step 4: Modelling – In this, used the following data science life cycle to have a detailed understanding of the final flow of the process,

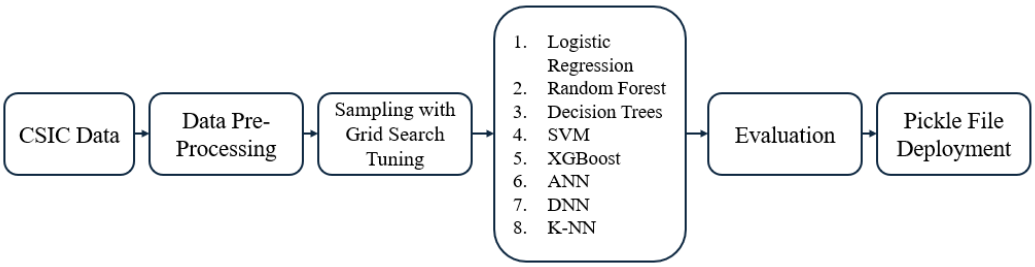


Figure 13: Data Science Life Cycle

Step 5: Evaluation – Different metrics like F1-Score, Accuracy, Recall, Precision, etc. to evaluate the best model and then deploy the same in the real-time cache partitioning environment.

Step 6: Deployment – The following is the scenario for the deployment. As seen in the flow diagram, we can find that,

a. Once the ML stage detects a suspicious cache, the cache partitioning is called
b. The output is shown as depicted in the next section

5.2 Python-based Cache Partitioning

```
class CacheLine:
    def __init__(self):
        self.tag = -1
        self.valid = False

class CacheSet:
    def __init__(self, associativity):
        self.lines = [CacheLine() for _ in range(associativity)]

class Cache:
    def __init__(self, cache_size, associativity):
        self.NUM_SETS = cache_size // associativity
        self.sets = [CacheSet(associativity) for _ in range(self.NUM_SETS)]
        self.partition_map = {}  # Map process ID to partition (way range)
        self.associativity = associativity

    def assign_partition(self, process_id, start_way, end_way):
        if start_way >= 0 and end_way < self.associativity and start_way <= end_way:
            self.partition_map[process_id] = (start_way, end_way)
        else:
            raise ValueError('Invalid partition range')

    def access(self, process_id, address):
        set_index = (address // self.associativity) % self.NUM_SETS
        tag = address // (self.NUM_SETS * self.associativity)
        if process_id not in self.partition_map:
            raise ValueError('Partition not assigned for process')

        start_way, end_way = self.partition_map[process_id]

        # Search for the tag in the assigned partition
        for way in range(start_way, end_way + 1):
```

```
            if self.sets[set_index].lines[way].valid and self.sets[set_index].lines[way].tag == tag:
                print(f"Cache hit in set {set_index}, way {way}")
                return True

        # Cache miss, replace a line in the assigned partition
        for way in range(start_way, end_way + 1):
            if not self.sets[set_index].lines[way].valid:
                self.sets[set_index].lines[way].tag = tag
                self.sets[set_index].lines[way].valid = True
                print(f"Cache miss, allocated in set {set_index}, way {way}")
                return False

        # If all lines in the partition are valid, replace the first one (simple replacement policy)
        self.sets[set_index].lines[start_way].tag = tag
        print(f"Cache miss, replaced in set {set_index}, way {start_way}")
        return False

# Example usage
cache_size = 16  # Total cache lines
associativity = 4  # Number of ways
cache = Cache(cache_size, associativity)

# Assign partitions to processes
cache.assign_partition(1, 0, 1)  # Process 1 gets ways 0-1
cache.assign_partition(2, 2, 3)  # Process 2 gets ways 2-3

# Simulate cache accesses
cache.access(1, 0)  # Process 1 accesses address 0
cache.access(2, 4)  # Process 2 accesses address 4
cache.access(1, 8)  # Process 1 accesses address 8
cache.access(2, 12)  # Process 2 accesses address 12
cache.access(1, 0)  # Process 1 accesses address 0 again (should be a hit)
```

Code Snippet 1: Cache partitioning

Explanation

a. The CacheLine class represents one single line in the cache which has a tag and a valid flag. The tag helps in identifying the data stored in the cache line while the valid indicates whether the cache line contains a valid data (true) or not (false)

b. The CacheSet class represents a set in a set-associative cache. This contains multiple CacheLine instances which determines by the cache's associativity. The associativity defines rge number of ways in each set

c. The Cache class represents the entire cache structure where the cache_size is the total number of cache lines. The associativity is the number of lines per set. NUM_SETS is the number calculated by dividing the total cache.

d. Assign_partition assigns a ran of ways to a specific process. This ensures the partition range is valis and updates partition_map.

e. Access simulates the cache access by a process which checks if the process has an assigned partition or not. It searches for a tag within the assigned partition. If found its cache hi if not found then the invalid line allocates a new data.

f. The output that is shown is as below way,

Cache miss, allocated in set 0, way 0: This indicates that the first access by Process 1 to address 0 resulted in a cache miss. A new cache line was allocated in set 0, way 0.
Cache miss, allocated in set 1, way 2: This indicates that the second access by Process 2 to address 4 resulted in a cache miss. A new cache line was allocated in set 1, way 2.
Cache miss, allocated in set 2, way 0: This indicates that the third access by Process 1 to address 8 resulted in a cache miss. A new cache line was allocated in set 2, way 0.
Cache miss, allocated in set 3, way 2: This indicates that the fourth access by Process 2 to address 12 resulted in a cache miss. A new cache line was allocated in set 3, way 2.
Cache hit in set 0, way 0: This indicates that the fifth access by Process 1 to address 0 resulted in a cache hit. The requested data was found in set 0, way 0.

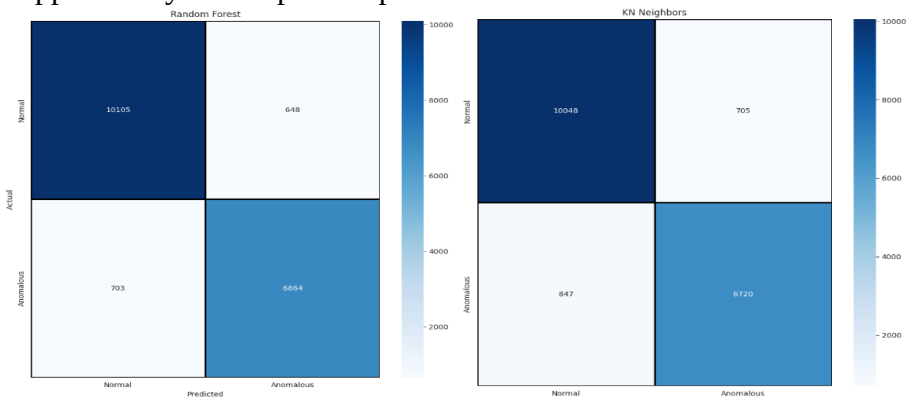
6. Evaluation

6.1 Stage 1: Machine Learning based vulnerability detection in the local cache

Table 1: Classification Report

Models	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Random Forest	93%	92%	92%	92%
KNN	92%	92%	92%	92%
Decision Tree	92%	92%	92%	92%
XGBM	92%	92%	92%	92%
Neural Network	89%	90%	89%	89%
SVC(Linear)	75%	75%	74%	74%
SGD	76%	76%	76%	76%
DNN	93%	92%	92%	92%

The table provides a comparative comparison of several machine learning models, including Random Forest, KNN, Decision Tree, XGBM, Neural Network, SVC (Linear), SGD, and DNN. The analysis focuses on their performance measures, namely accuracy, precision, recall, and F1 score. According to the data, Random Forest and DNN are the most successful models, attaining the highest scores in all criteria at 93%. These models are deemed to be the most dependable and efficient for the provided dataset. KNN, Decision Tree, and XGBM had impressive scores of 92% in all assessed parameters, demonstrating their robust performance and possible applicability to comparable problems.



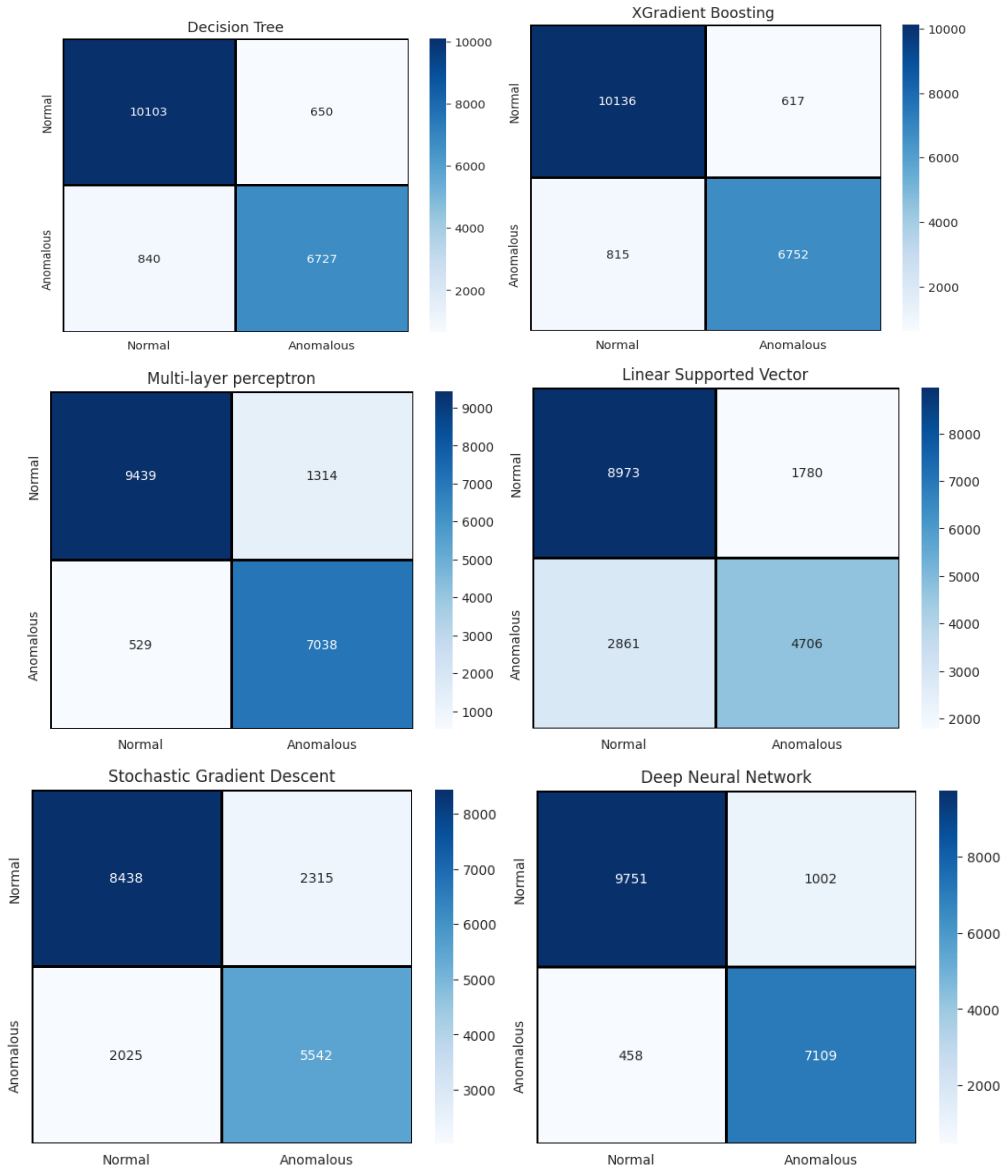


Figure 14: Confusion matrices for all the models. (a) Random Forest (b) k-NN (c) Decision Trees (d) XGBoost (e) Multi Perceptron (f) SVC (g) SGD (h) DNN

The Neural Network demonstrates a respectable level of performance with an accuracy of 89%, but, it falls short of the highest-performing models. At the bottom end of the performance spectrum, SVC (Linear) and SGD models demonstrate noticeably less efficacy, with their scores consistently around 75-76% for all criteria. This suggests that these models are generally ill-suited for the given dataset. To summarize, Random Forest and DNN exhibit the highest level of proficiency among the models, while KNN, Decision Tree, and XGBM also display strong performance. The Neural Network is effective, while SVC (Linear) and SGD are less suited for this specific examination. From the above confusion matrix, we can find that random forest has predicted correctly the suspicious caches. The ranking of the models can be checked in the following,

Rank 1: top 3-Models Majority voter - MAE: 0.0734 - Error: 7.34%
Rank 2: Random Forest - MAE: 0.0737 - Error: 7.37%
Rank 3: top 5-Models Majority voter - MAE: 0.0747 - Error: 7.47%
Rank 4: Gradient Boosting - MAE: 0.0782 - Error: 7.82%

Rank 5: Deep Neural Network - MAE: 0.0797 - Error: 7.97%
Rank 6: Decision Tree - MAE: 0.0813 - Error: 8.13%
Rank 7: K-Nearest Neighbors - MAE: 0.0847 - Error: 8.47%
Rank 8: Multi-Layer Perceptron - MAE: 0.1006 - Error: 10.06%
Rank 9: Stochastic Gradient Descent - MAE: 0.2369 - Error: 23.69%
Rank 10: Support Vector Machine - MAE: 0.2533 - Error: 25.33%

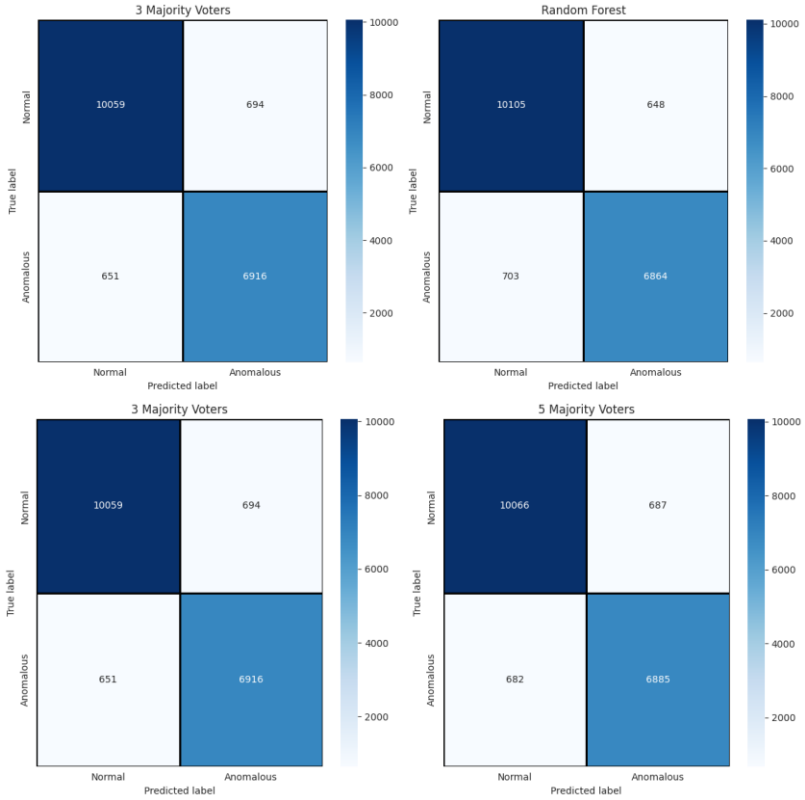


Figure 15: Majority Voters – Ensemble model confusion matrix

6.2 Stage 2 Cache Partitioning

Case Study 1: AI-based cache partitioning using Suspicious Cache

Figure 16: Cache Partitioning in real-time using a response Front-end

In this have used a streamlit frontend for the deployment of the solution. When the ML algorithm detects the suspicious Cache, then the solution will be deployed else the solution is devoid of the deployment. As can be seen when the suspicious data is chosen, the probability rate is 84% which is more than the threshold of 64%. i.e. if the probability is more than the threshold it is suspicious, else not. In case of suspiciousness, we can see at the right bottom the partition code is executed and the partition has happened. On the top right side, the details of the suspicious are shown. We can find the important features based on the gradient boosting model. In the middle section, we can change the cache address to see if the partition is hit or not.

6.4 Case Study 2: AI-based cache partitioning using Normal or Non – Suspicious Cache

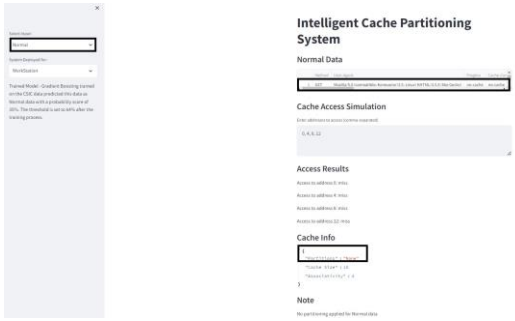


Figure 17: Cache Partitioning in real-time using a response Front end with normal Cache
The highlighted part shows that the probability is found to be 35% which is less than the threshold set that is 64% for it being categorized into the Normal class. The table on the right-hand side discusses the content of the data. While the right-side bottom says that no partition has happened.

7: Conclusion and Future Work

Finally, different machine learning models exhibit different patterns of performance as measured by F1 score, recall, accuracy, and precision. The consistency of Random Forest and DNN in reaching 93% across all assessment criteria highlights their efficiency and dependability for the dataset in issue, positioning them as the top performers. With a combined score of 92% across all parameters, KNN, Decision Tree, and XGBoost follow closely behind, all suggesting strong performance appropriate for comparable jobs. Neural Network shows efficacy with an accuracy rate of 89%. On the other hand, SGD and SVC (Linear) do less well, scoring between 75% and 76%, which indicates that they aren't very well-suited to the needs of the dataset.
By breaking out the models' prediction strengths and shortcomings, the confusion matrices provide a more nuanced picture of their performance. When a balance between accuracy and precision recall is of the utmost importance, XGBoost, Random Forest, and DNN are the best options because to their constant and thorough performance. Due to their relatively lower performance, methods such as SVC (Linear) and SGD may not be the best choice for jobs that prioritize computational efficiency or linear separability. This research highlights the significance of using machine learning models that are customized to unique dataset properties and job needs to maximize the accuracy of predictions. Also while coming to the picture of deploying the best model, it should be light with less computation requirement and space, and in these scenarios, XgBoost qualifies.

The machine learning models tested over the last few months have yielded insightful information. While the evaluation conducted thus far is based on sample data, these models must be applied to real-world datasets to be more effective and to increase the solution's generalisability. We will be able to evaluate this response system's effectiveness in real-time scenarios—especially when real vulnerabilities are found—by integrating it with other software. Furthermore, investigating and implementing additional algorithms and large language models (LLMs) may enhance the system's resilience and functionality.

1 References

Alaca, Y. and Çelik, Y., 2023. Cyber attack detection with QR code images using lightweight deep learning models. Computers & Security, 126, p.103065.

Alsaade, F.W. and Al-Adhaileh, M.H., 2023. Cyber attack detection for self-driving vehicle networks using deep autoencoder algorithms. Sensors, 23(8), p.4086.

Alzahrani, A.O. and Alenazi, M.J., 2023. ML-IDSDN: Machine learning-based intrusion detection system for software-defined network. Concurrency and Computation: Practice and Experience, 35(1), p.e7438.

Basholli, F., Daberdini, A. and Basholli, A., 2023. Detection and prevention of intrusions into computer systems. Advanced Engineering Days (AED), 6, pp.138-141.Berrar, D., 2019. Cross-validation.

Chen, C., Li, Y., Wang, Q., Yang, X., Wang, X. and Yang, L.T., 2023. An Intelligent Edge-Cloud Collaborative Framework for Communication Security in Distributed Cyber-Physical Systems. IEEE Network.

Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree-boosting system. In Proceedings of the 22nd acm sigkdd International Conference on knowledge discovery and data mining (pp. 785-794).

De Ville, B., 2013. Decision trees. Wiley Interdisciplinary Reviews: Computational Statistics, 5(6), pp.448-455.

Dhanya, K.A., Vajipayajula, S., Srinivasan, K., Tibrewal, A., Kumar, T.S. and Kumar, T.G., 2023. Detection of network attacks using machine learning and deep learning models. Procedia Computer Science, 218, pp.57-66.

Ding, C., Wang, S., Wang, Y., Wu, Z., Sun, J. and Mao, Y., 2023. Machine-learning-based detection for quantum hacking attacks on continuous-variable quantum-key-distribution systems. Physical Review A, 107(6), p.062422.

Fang, J., Nie, Z. and Zhao, L.A., 2022, March. PACP: A prefetch-aware multi-core shared cache partitioning strategy. In Proceedings of the 8th International Conference on Computing and Artificial Intelligence (pp. 246-251).

Grossberg, S., 2013. Recurrent neural networks. Scholarpedia, 8(2), p.1888. Holtryd, N.R., Manivannan, M. and Stenström, P., 2023, May. SCALE: Secure and Scalable Cache Partitioning. In 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) (pp. 68-79). IEEE. Kelleher, J.D., 2019. Deep learning. MIT press.

Ketkar, N. and Ketkar, N., 2017. Stochastic gradient descent. Deep learning with Python: A hands-on introduction, pp.113-132.

Krishnan, P., Jain, K., Aldweesh, A., Prabu, P. and Buyya, R., 2023. OpenStackDP: a scalable network security framework for SDN-based OpenStack cloud infrastructure. Journal of Cloud Computing, 12(1), p.26.

Lin, C., Xu, Y., Fang, Y. and Liu, Z., 2023. VulEye: A novel graph neural network vulnerability detection approach for PHP application. Applied Sciences, 13(2), p.825.

Liu, W., Cui, J., Li, T., Liu, J. and Yang, L.T., 2022. A space-efficient fair cache scheme based on machine learning for nvme ssds. IEEE Transactions on Parallel and Distributed Systems, 34(1), pp.383-399.

park, J., Yeom, H. and Son, Y., 2020. Page reusability-based cache partitioning for multi-core systems. IEEE Transactions on Computers, 69(6), pp.812-818.

Peterson, L.E., 2009. K-nearest neighbor. Scholarpedia, 4(2), p.1883.

Popescu, M.C., Balas, V.E., Perescu-Popescu, L. and Mastorakis, N., 2009. Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems, 8(7), pp.579-588.

Pujitha, K., Nandini, G., Sree, K.T., Nandini, B. and Radhika, D., 2023, May. Cyber hacking breaches prediction and detection using machine learning. In 2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN) (pp. 1-6). IEEE.

Purba, M.D., Ghosh, A., Radford, B.J. and Chu, B., 2023, October. Software vulnerability detection using large language models. In 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 112-119). IEEE.

Qiu, J., Hua, Z., Liu, L., Cao, M. and Chen, D., 2022. Machine-learning-based cache partition method in cloud environment. Peer-to-Peer Networking and Applications, 15(1), pp.149-162.

Rigatti, S.J., 2017. Random forest. Journal of Insurance Medicine, 47(1), pp.31-39.

Schorfheide, F. and Wolpin, K.I., 2012. On the use of holdout samples for model selection. American Economic Review, 102(3), pp.477-481.

Shang, Y., 2024. Detection and Prevention of Cyber Defense Attacks using Machine Learning Algorithms. Scalable Computing: Practice and Experience, 25(2), pp.760-769.

Stutz, D., de Assis, J.T., Laghari, A.A., Khan, A.A., Deshpande, A., Kulkarni, D., Terziev, A., de Jesus, M.A. and Grata, E.G., 2024. Cyber Threat Detection and Mitigation Using Artificial Intelligence–A Cyber-physical Perspective. Applying Artificial Intelligence in Cybersecurity Analytics and Cyber Threat Detection, pp.107-133.

Sun, H., Sun, C., Tong, H., Yue, Y. and Qin, X., 2024. A Machine Learning-empowered Cache Management Scheme for High-Performance SSDs. IEEE Transactions on Computers.

Vartouni, A.M., Kashi, S.S. and Teshnehlab, M., 2018, February. An anomaly detection method to detect web attacks using stacked auto-encoder. In 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS) (pp. 131-134). IEEE.

Wang, H. and Hu, D., 2005, October. Comparison of SVM and LS-SVM for regression. In 2005 International conference on neural networks and brain (Vol. 1, pp. 279-283). IEEE.

Wang, P., Liu, Y., Liu, Z., Zhao, Z., Liu, K., Zhou, K. and Huang, Z., 2024. $\varepsilon $-LAP: A Lightweight and Adaptive Cache Partitioning Scheme with Prudent Resizing Decisions for Content Delivery Networks. IEEE Transactions on Cloud Computing.

Wang, Y., Yang, J. and Wang, Z., 2023. Multi-tenant in-memory key-value cache partitioning using efficient random sampling-based LRU model. IEEE Transactions on Cloud Computing.

Wang, Z. and O'Boyle, M.F., 2010, September. Partitioning streaming parallelism for multi-cores: a machine learning based approach. In Proceedings of the 19th International Conference on Parallel Architectures and compilation techniques (pp. 307-318).

Zhang, C., Liu, H., Zeng, J., Yang, K., Li, Y. and Li, H., 2024, April. Prompt-enhanced software vulnerability detection using ChatGPT. In Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (pp. 276-277)

Zhao, Y., Xiao, A., Wu, S., Jiang, C., Kuang, L. and Shi, Y., 2023, August. Content-Adaptive Cache Partitioning for Two-Layer Mobile Edge Networks. In 2023 IEEE/CIC International Conference on Communications in China (ICCC) (pp. 1-6). IEEE.

Zhao, Y., Xiao, A., Wu, S., Jiang, C., Kuang, L. and Shi, Y., 2024. Adaptive Partitioning and Placement for Two-Layer Collaborative Caching in Mobile Edge Computing Networks. IEEE Transactions on Wireless Communications.