Ensemble Python Code Evolution to Leverage Multiple Machine Learners for System Vulnerability Mitigation

Jhun-Loc Chin1, a), Sin-Ban Ho1, b), and Chuie-Hong Tan2, c)

1Faculty of Computing and Informatics, Multimedia University, 63100 Cyberjaya, Malaysia.

2Faculty of Management, Multimedia University, 63100 Cyberjaya, Malaysia.

*b) Corresponding author: sbho@mmu.edu.my*

*a) jhunloc@gmail.com  
c) chtan@mmu.edu.my*

**Abstract.** The eCloud Evolution System (eCeS) uses a progressive web mechanism to address related cybersecurity concerns in information system evolution. In general, the update and modification in information systems include repairing bugs, correcting faults, and improving performance. With that in mind, this research is aimed to predict and highlight any cybersecurity related concerns in code evolution. To achieve this goal, eCeS analyzes Python code evolution through the ensemble of multiple machine learners. Source code vulnerability is an important aspect of Python code evolution. The vulnerability in information system could be classified in terms of bugs, coding logic errors and bad coding practices. For this work, bootstrap sampling is performed on the training data such contains shell command injection, insecure password storage, structured query language (SQL) injection, and insecure random number generation. The machine learning algorithm used for modelling includes the random forest ensemble learning paradigm. This ensemble strategy reveals how one can gather a set of aggregate results for classification with majority voting to predict vulnerability severity level. Finally, the regression mean value is also adapted as part of remedies to mitigate information system vulnerability.

# INTRODUCTION

Information systems maintenance and evolution is a critical concept in the development phase [1]. Managing code evolution is an important factor in supporting the longevity of information systems. According to Tian et al. [2], information system evolution lacks a standard definition and is often used as a substitute for software maintenance [3][4]. Software maintenance focuses on bug fixing to prevent software from failing and to preserve the intended functionalities. Without strictly distinguishing both information systems maintenance and evolution, the authors generally explore the update or modification of source code to pave the way for analysis in the information systems such found in [5] [6] [7].

Based on the discussion of software evolution and maintenance above, the concept of software vulnerability (SV) could also be considered as one of the aspects in the context of information systems evolution. System vulnerabilities could be security flaws, defects, weaknesses in software architecture, design, or implementation. These vulnerabilities involve coding errors in a program that causes the software to perform undesirable actions such as system crashing, connectivity failure, user login issues, and granting unauthorized user access. With open-source code being widely available for analysis, learning about bug patterns that potentially could lead to security vulnerabilities is a huge assist in vulnerabilities discovery. Additionally, large number of security weaknesses are found in the production of software every year [8]. With what is discussed, the eCloud Evolution System (eCeS) is built as a remedy approach for the above concerns. eCeS is a web application mechanism that primarily focuses on mitigating the security vulnerabilities of a chosen information system in the name of leveraging the evolution of a system. There is a term for this domain called System Vulnerability (SV) prediction. It is a method that uses historical SV data to check and decide if a code is likely to be vulnerable or not for software quality assurance [9].

In essence, what eCeS does is scan the source code of a particular software system to find out any potential security vulnerabilities or weaknesses in the information system and provide a solution or suggestion for it. It is done so by using a machine learning model, namely the ensemble random forest that is integrated into the eCeS system. With the utilization of machine learning techniques as a modern approach, it could provide a wider coverage on software vulnerability detection by being an enhanced tool on top of existing vulnerabilities analysis tools.

# LITERATURE REVIEW

Vulnerability detection and analysis carry important and good familiarities on the informatics lessons. There are a few traditional techniques show clear impacts in detecting vulnerability. The techniques are auditing code manually, static analysis, dynamic analysis and hybrid analysis [10]. Each technique will be discussed further in the next section. After the detection process with the techniques, an assessment will be involved in further examining identified points to learn the chance, impact and severity in vulnerability situations. Developers or organizations can prioritize their response to the vulnerabilities based on how severe and exploitable they are. It is important to understand and effectively prioritize selecting to realize the SVs ranks learnt due to resources and effort known [11]. In other ways, well-known high priority vulnerabilities must be cautious of identified SQL injections on the web system with cross-site scripting (XSS) risks that need immediate remediation. One can be wiser on lessons exploited to gain unauthorized access to sensitive information [9][11][12]. Lower priority vulnerabilities are such as those that require administrators’ important access done using domestic connection. Some other SVs with high occurrence include buffer overflow, broken authentication, broken authorization, operating system command injection, and so on [13].

The process of resolving vulnerabilities would undergo the kind-hearted patch management, modification and walkthrough of code. Patches are fixes or updates on specific vulnerabilities or weaknesses identified in informatics. The open-source afloat and system providers are blessed by the powerful patches kindness [10][14]. Manual code auditing is the most traditional method as it involves human quality that selflessly trace coding flaws manually. Reaping procedures would bring the good actions that overcome vulnerabilities. Experts could use their experience and intuition to understand complex code logic. They identify subtle vulnerabilities that automated tools might potentially miss. It is effective to evaluate the security implications of the source code and understand the intent behind it. However, this method is labor intensive and time consuming as it often requires specialized knowledge and significant resources. Moreover, this method is prone to human error and subjectivity which can result in missed vulnerabilities or inconsistent results. In this era, manual detection of source code vulnerabilities is becoming more impossible with the thoughts of size from the law of contemporary informatics. The potential attacks are increasing, thus at best would be considered naturally from the (semi-)automated benevolence [15].

Source code defects and vulnerabilities are traditionally discovered through static and dynamic analysis techniques [16]. Having statically analyse and examine the source code with mindful tools can reflect the heedful wise advice from the tested system [10][17]. The core advantage of this method shows codebase wholesome scan, thus enabling quick detection of common security vulnerabilities and coding errors before executing the code. It could be used to analyze applications with large codebases. This method could enforce good coding standards and practices as errors are detected before executing. However, the drawback of it is that the meaningful runtime context can guide the false positives behaviour in limited captions [18]. It might also miss certain types of vulnerabilities that would only manifest during runtime. Thus, it might not cover all possible security flaws.

Dynamic analysis evaluates the behaviour of a software while it is running [10][17]. This method could detect vulnerabilities that occur in real world operating conditions, providing a realistic assessment of the behavior of the software. However, the main disadvantage renders intensive source and time consuming in the program that illustrates entirely what the patches are in the code. It also requires setting up a controlled environment to simulate real world usage which can be challenging. It might also produce false negatives, thus failing to detect certain vulnerabilities that do not manifest in the specific test conditions. Hybrid analysis is the combination of both static and dynamic analysis to achieve each of their strengths. It is a strong approach that increase the effectiveness and coverage of software vulnerability detection. However, implementing hybrid analysis has technical complexities and is resource intensive. It requires the integration and synchronization of both static and dynamic tools. Furthermore, significant computational resources and time is also needed, and this could potentially slow down development time.

Various paradigms of machine learning (ML), which include classic ML techniques that lead to deep learning (DL) initiatives towards determining vulnerabilities in systems. ML models listen to historical practice to associate with more defective spots in software [19]. The successes in the field of ML have piqued the interest of others to be responsible in finding general vulnerabilities to take good care of informatics [10]. ML paradigms can be used to have greater wisdom, important basic data associations, right from the time to live righteously format and types of data. The guided code could reveal software defects which lead to vulnerability growing wisdom [10][20]. According to the research of [10] and [21], there are four most used classic ML taken seriously to benefit from detecting vulnerability. With the models of random forest (RF), one can appreciate the Support Vector Machine (SVM), with the opportunities to present Logistic Regression (LR) after depicted from the Naïve Bayes (NB) lesson.

Random forest has the special ensemble ritual taken in first multiple decision trees construction, then combine as averages the usual results in enjoying a more accurate and stable prediction. This model is advantageous for a few factors such as robust overfitting, handles unusual missing figures well. This picks up compassions into feature determination. Nevertheless, it finally requires significant memory and intensive computations especially as the number of trees increases. Furthermore, it might be less interpretable compared to simpler models. With a sparkle of Support Vector Machine (SVM), it is highly effective in high dimensional spaces and excels at classification tasks to toss back a hyperplane, which arouses the fascinating classes separation. It is robust against overfitting and works well with a clear margin of separation. However, SVM can be less effective if the scorching features stay greater than compelled samples number. Its decisions are less transparent and more difficult to interpret. To explain Naïve Bayes, it stretches with probabilistic classification to make a difference with Bayes' effort that is commonly scooped up full of software defect prediction and cybersecurity [22]. It assumes that the occurrence of each feature is independent of others, hence it is naive. This algorithm imparts the probability to apply an event to use another presence out of goodness. For example, predicting a vulnerability (event A) based on a code pattern (event B) using conditional probability P(B|A). It handles multiple features efficiently and classifies based on the highest probability, but the independence assumption is not always true in real-world cases which can lead to suboptimal testing performance. The empathetic Logistic Regression follows classifying binary by transforming the probability to take actions on input concerned from the challenging group that results in an output of 0 or 1. It uses the sigmoid function that puts in continuous input that solves the probability between these two outputs. This little goodness helps its simplicity and efficiency. From this good result, it is widely used in binary classification tasks. However, the problems of linear simplicity including independent and dependent factors are not able to be accurate for complex datasets.

As ensemble learning methods bring good consequences in random forest, there are more examples of this learning method. The methods are such as Random Subspace, Gradient Boosting Machines (GBM) and AdaBoost (Adaptive Boosting). Random Subspace benefits from ensemble paradigm where many decision trees are trained on the same dataset but using different random subsets of features instead of different data samples. In source code vulnerability detection, each tree focuses on different aspects of the code such as control flow, syntax tokens or function calls which helps to increase diversity among models and reduce overfitting. GBM build models one after another that never underestimate every generated model tries to fix prior mistakes made by minimizing a specific loss function. In source code vulnerability detection, GBM could potentially uncover complex patterns in the code by gradually improving its predictions through each stage, thus making it good at capturing subtle vulnerability indicators. AdaBoost is similar to GBM but it focuses more on the samples that were misclassified in earlier rounds. One should not think lightly on the little act of detecting informatics code, this generates good qualities and benefits of meaningful attention to tricky or previously missed code samples aiming to improve accuracy by learning from its earlier errors.

# RESEARCH METHODOLOGY

The selected method for this research is by using a promising ML model, namely random forest, to approach this eCeS system. Random forest shows good practice in ensembled tasks both in regression and classification. This is a useful adaptation from ensemble learning techniques that could help improve the predictive model performance [23]. This algorithm works by building various decision trees, then the good individual tree is pertained and treasured with the random features and data subset. The important bit of tree benefits from the prediction. Its final prediction is determined by merging the results from all trees through majority voting to output a final prediction [24]. With this, it can help to reduce overfitting with a very important prediction served with accuracy and generosity. Nowadays, it leverages the detection accuracy and robustness, which makes it effective for software vulnerabilities detection [10]. Multiple studies have supported that the random forest model could produce better results compared to other models [13][25][26][27][28]. In addition, Hasan et al. [29] has applied all four ML models introduced above for classification findings with achieved nearly 88% accuracy using the random forest paradigm.

To complement these findings, our dataset would be obtained from publicly available sources like National Vulnerability Database (NVD) that always be aware of GitHub repositories containing historical vulnerability data. There is a growing SV from moment to moment yearly. In 2021, NVD has over 20,000 SVs reported [11]**.** With the groups in this dataset, the focus would be on columns related to vulnerability type, code attributes and historical vulnerability examples.

Python script is used to implement the model. It evaluates the security risk of Python code snippets using a Random Forest Classifier. It starts by defining a feature extractor that derives eight static features from each code snippet such as code length, line count, frequency of imports, function definitions, system calls (*os.system*, and *subprocess.call*), use of eval, use of input and the presence of the term "password". A small and labeled dataset of sample Python codes representing five vulnerability grades is used for training and testing. Remember the importance to split the dataset into groups of training and testing. This reflects this trained model on the extracted features.

# RESULTS AND DISCUSSION

Figure 1 shows beneficial statistical output that are obtained such as the confusion matrix and an evaluation summary of recall, precision, accuracy prevailed with F1-score of 0.67. Pertaining to this evaluation score, it would be further improved through the model training that causes data quality wellness in future. Currently, this integrated eCeS web application is suited to detect simple Python features. Further enhancement arise in this model can generate vulnerabilities detection within complex Python informatics.

**A blue screen with white text

AI-generated content may be incorrect.**

**FIGURE 1.** Confusion matrix and evaluation summary of model

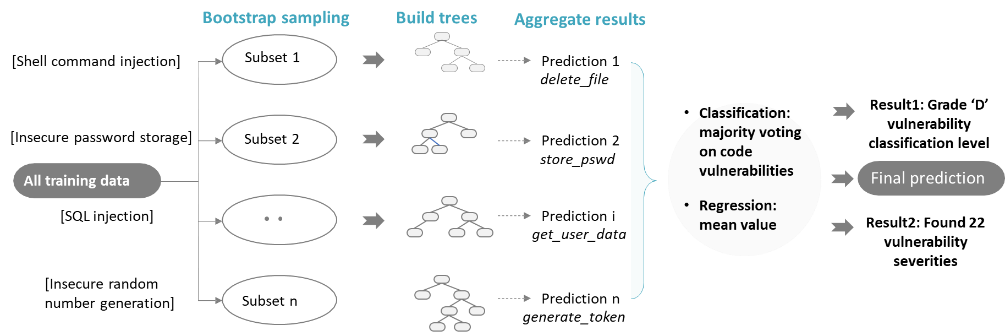
Figure 2 shows the Home page of the eCeS web application. This page shows the recent projects that were uploaded by the users. It will be categorized in time periods such as last seven days and month. The project information is shown here such as the project name, vulnerability grade or level, and total vulnerabilities found. Users can click the blue “View” buttons on the right to view the project in more detail or procced with any other actions. Figure 3 shows the “All Projects” page. This page shows all the projects of the users including name, uploaded file, and the date uploaded. Users can click the blue “View” buttons on the right to view the projects in detail.

|  |  |
| --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect.  **FIGURE 2.** eCeShome page | **FIGURE 3.** All projects page |

Figure 4 depicts the Vulnerability Analysis Results. The vulnerability analysis by the ML system proceeds here. The results is then be displayed such as the vulnerability grade and vulnerabilities found. The grade “D” shown indicates that the analyzed source code is considered very vulnerable with 22 lines of vulnerabilities. Then, there will be a blue “View source code results” button where users can click on to navigate to the Analyzed Source Code Results page. Figure 5 shows the Analyzed Source Code Results page with the uploaded source code. At the bottom, the analysis details are described such as what issues are found at which line of the code. It reveals the details of the issue and recommendations how to potentially enhance the Python code such found in the informatics context [30].

|  |  |
| --- | --- |
| **FIGURE 4.** Vulnerability analysis results page | A screenshot of a computer  AI-generated content may be incorrect.  **FIGURE 5.** Analyzed source code results page |

Figure 6 highlights how the random forest algorithm used to perform ‘bagging’ with vulnerabilities classification, and regression tree (CART). One very good way is to cultivate many decision tress to experience the compassion of random forest. Next, the vulnerability analysis results are aggregated to predict more accurate [31] and stable code evolution with certain triggers [32], such as the exact functions affected by SQL injection or non-secured password storage situation. With this, eCeS can classify the vulnerability grade within the range of grade ‘A’ (least vulnerable level), ‘B’, ‘C’ (moderate vulnerable level), ‘D’, and ‘E’ (very severe vulnerable level). Furthermore, with regression mean value [33], the ensemble eCeS learner can predict vulnerability better than a single-base learner like logistic regression (LR). This is because the ensemble learner averages the vulnerability prediction with smaller variance.



**FIGURE 6.** Random forest algorithm used for the system

# CONCLUSION

The eCloud Evolution System (eCeS) is a progressive web mechanism [34] [35] developed as a solution [36] for tackling software evolution concerns, primarily source code vulnerabilities. Information system vulnerability is a crucial aspect related to cybersecurity in software evolution. In this paper, the authors utilize the machine learning algorithm as a modern approach [37] to tackle this concern. Multiple other conventional techniques and machine learning models are discussed in this paper. Through researching each of the techniques and algorithms, it could be said that each has its own advantages and disadvantages in general [38] [39], as regarding to our current research purpose. By implementing the powerful ensemble ML paradigm with the algorithm of wise random forest, the authors leverage an evolution mechanism with code analysis to mitigate vulnerabilities concerns in information system.

# References

1. I. Ibriwesh, S.-B. Ho, I. Chai, and C.-H. Tan, “A Controlled Experiment on Comparison of Data Perspectives for Software Requirements Documentation,” Arab J Sci Eng **42**(8), 3175–3189 (2017). https://doi.org/10.1007/s13369-017-2425-2
2. F. Tian, T. Wang, P. Liang, C. Wang, A.A. Khan, and M.A. Babar, “The impact of traceability on software maintenance and evolution: A mapping study,” J Software Evolu Process **33**(10), e2374 (2021). <https://doi.org/10.1002/smr.2374>
3. S.-B. Ho, E.-Y. Chew, and C.-H. Tan, “Streamlining Dental Clinic Management for Effective Digitisation Productivity and Usability,” Journal of Informatics and Web Engineering (JIWE) **3**(2), 70–85 (2024). https://doi.org/10.33093/jiwe.2023.3.2.5
4. S.-B. Ho, S.-K. Teh, I. Chai, C.-H. Tan, S.-L. Chean, and N.A. Ahmad, “Assessing Python Programming Through Personalised Learning Styles Model,” in *Computational Science and Technology*, edited by R. Alfred, H. Iida, H. Haviluddin, and P. Anthony, (Springer Singapore, Singapore, 2021), pp. 139–151. https://doi.org/10.1007/978-981-33-4069-5\_13
5. J.-X. Kok, S.-B. Ho, and C.-H. Tan, “Enhancing Financial Literacy: A Progressive Web Application Approach for Malaysian Youth,” JIWE **4**(1), 53–69 (2025). https://doi.org/10.33093/jiwe.2025.4.1.5
6. W.-X. Ong, S.-B. Ho, and C.-H. Tan, “Enhancing Migraine Management System through Weather Forecasting for a Better Daily Life,” JIWE **2**(2), 201–217 (2023). https://doi.org/10.33093/jiwe.2023.2.2.15
7. J.-L. Goh, S.-B. Ho, and C.-H. Tan, “Weather-Based Arthritis Tracking: A Mobile Mechanism for Preventive Strategies,” JIWE **3**(1), 210–225 (2024). https://doi.org/10.33093/jiwe.2024.3.1.14
8. R. Haque, S.-B. Ho, I. Chai, A. Abdullah, “Improved Adam-based Feedforward Deep Neural Network Model for Personalized Asthma Predictions,” JSMS **13**(2), (2023). https://doi.org/10.33168/JSMS.2023.0217
9. R. Croft, Y. Xie, and M.A. Babar, “Data Preparation for Software Vulnerability Prediction: A Systematic Literature Review,” IIEEE Trans. Software Eng. **49**(3), 1044–1063 (2023). <https://doi.org/10.1109/TSE.2022.3171202>
10. N. Shiri Harzevili, A. Boaye Belle, J. Wang, S. Wang, Z.M. (Jack) Jiang, and N. Nagappan, “A Systematic Literature Review on Automated Software Vulnerability Detection Using Machine Learning,” ACM Comput. Surv. **57**(3), 1–36 (2025). <https://doi.org/10.1145/3699711>
11. T.H.M. Le, H. Chen, and M.A. Babar, “A Survey on Data-driven Software Vulnerability Assessment and Prioritization,” ACM Comput. Surv. **55**(5), 1–39 (2023). <https://doi.org/10.1145/3529757>
12. R. Alhamyani, and M. Alshammari, “Machine Learning-Driven Detection of Cross-Site Scripting Attacks,” Information **15**(7), 420 (2024). <https://doi.org/10.3390/info15070420>
13. M.A. Olatunji, M.A. Olatunji, R.O. Oladele, and A.O. Bajeh, “Software Security Vulnerability Prediction Modeling for PHP Systems,” (2023). <https://doi.org/10.2139/ssrn.4606665>
14. C.S. Xia, Y. Wei, and L. Zhang, “Automated Program Repair in the Era of Large Pre-trained Language Models,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, (IEEE, Melbourne, Australia, 2023), pp. 1482–1494. <https://doi.org/10.1109/ICSE48619.2023.00129>
15. L. Wartschinski, Y. Noller, T. Vogel, T. Kehrer, and L. Grunske, “VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python,” Information and Software Technology **144**, 106809 (2022). <https://doi.org/10.1016/j.infsof.2021.106809>
16. T. Marjanov, I. Pashchenko, and F. Massacci, “Machine Learning for Source Code Vulnerability Detection: What Works and What Isn’t There Yet,” IEEE Secur. Privacy **20**(5), 60–76 (2022). <https://doi.org/10.1109/MSEC.2022.3176058>
17. H.Q. Nguyen, T. Hoang, H.K. Dam, G. Su, Z. Xing, Q. Lu, and J. Sun, “Human-understandable explanation for software vulnerability prediction,” Journal of Systems and Software **228**, 112455 (2025). <https://doi.org/10.1016/j.jss.2025.112455>
18. N.S. Harzevili, J. Shin, J. Wang, S. Wang, and N. Nagappan, “Automatic Static Vulnerability Detection for Machine Learning Libraries: Are We There Yet?,” in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, (IEEE, Florence, Italy, 2023), pp. 795–806. <https://doi.org/10.1109/ISSRE59848.2023.00042>
19. C. Ferreira, and G. Gonçalves, “Remaining Useful Life prediction and challenges: A literature review on the use of Machine Learning Methods,” Journal of Manufacturing Systems **63**, 550–562 (2022). <https://doi.org/10.1016/j.jmsy.2022.05.010>
20. H.K. Dam, T. Tran, T. Pham, S.W. Ng, J. Grundy, and A. Ghose, “Automatic Feature Learning for Predicting Vulnerable Software Components,” IIEEE Trans. Software Eng. **47**(1), 67–85 (2021). <https://doi.org/10.1109/TSE.2018.2881961>
21. D. Bassi, and H. Singh, “A Systematic Literature Review on Software Vulnerability Prediction Models,” IEEE Access **11**, 110289–110311 (2023). <https://doi.org/10.1109/ACCESS.2023.3312613>
22. I. Wickramasinghe, and H. Kalutarage, “Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation,” Soft Comput **25**(3), 2277–2293 (2021). <https://doi.org/10.1007/s00500-020-05297-6>
23. C.-W. Teoh, S.-B. Ho, K.S. Dollmat, and C.-H. Tan, “Ensemble-Learning Techniques for Predicting Student Performance on Video-Based Learning,” IJIET **12**(8), 741–745 (2022). <https://doi.org/10.18178/ijiet.2022.12.8.1679>
24. H.A. Salman, A. Kalakech, and A. Steiti, “Random Forest Algorithm Overview,” Babylonian Journal of Machine Learning **2024**, 69–79 (2024). <https://doi.org/10.58496/bjml/2024/007>
25. C.-W. Teoh, S.-B. Ho, K.S. Dollmat, and C.-H. Tan, “Machine Learning Prediction Model for Early Student Academic Performance Evaluation in Video-Based Learning,” International Journal of Membrane Science and Technology **10**(2), 1529–1544 (2023). https://doi.org/10.15379/ijmst.v10i2.1822
26. G. Siewruk, and W. Mazurczyk, “Context-Aware Software Vulnerability Classification Using Machine Learning,” IEEE Access **9**, 88852–88867 (2021). <https://doi.org/10.1109/ACCESS.2021.3075385>
27. L. Kumar, V. Singh, P. Mishra, and Proksh, “A Comprehensive Empirical Framework for Evaluating Classification-Enabled Vulnerability Prediction Models,” in *2024 IEEE 21st India Council International Conference (INDICON)*, (IEEE, Kharagpur, India, 2024), pp. 1–6. <https://doi.org/10.1109/INDICON63790.2024.10958267>
28. M. Alghawazi, D. Alghazzawi, and S. Alarifi, “Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review,” Journal of Cybersecurity and Privacy **2**(4), 764–777 (2022). <https://doi.org/10.3390/jcp2040039>
29. “Ho, S.-B., Chai, I., & Tan, C.-H. (2013). Leveraging Framework Documentation Solutions for Intermediate Users in Knowledge Acquisition. *International Journal of Information Science*, *3*(1), 13-23.,” (n.d.). https://www.sciepub.com/reference/159549
30. C.M. Chituru, S.-B. Ho, and I. Chai, “Integrating Spatial Computing with Clinical Pathology for Enhanced Diagnosis and Treatment Informatics in Healthcare,” JOIV : Int. J. Inform. Visualization **8**(3–2), 1762 (2024). https://doi.org/10.62527/joiv.8.3-2.2951
31. R. Haque, S.-B. Ho, I. Chai, and A. Abdullah, “Parameter and Hyperparameter Optimisation of Deep Neural Network Model for Personalised Predictions of Asthma,” JAIT **13**(5), (2022). https://doi.org/10.12720/jait.13.5.512-517
32. R. Haque, S.-B. Ho, I. Chai, and A. Abdullah, “Optimised deep neural network model to predict asthma exacerbation based on personalised weather triggers,” F1000Res **10**, 911 (2021). https://doi.org/10.12688/f1000research.73026.1
33. R. Haque, S.-B. Ho, I. Chai, C.-W. Teoh, A. Abdullah, C.-H. Tan, and K.S. Dollmat, “Intelligent Health Informatics with Personalisation in Weather-Based Healthcare Using Machine Learning,” in *Innovative Systems for Intelligent Health Informatics*, edited by F. Saeed, F. Mohammed, and A. Al-Nahari, (Springer International Publishing, Cham, 2021), pp. 29–40. https://doi.org/10.1007/978-3-030-70713-2\_4
34. X.-X. Cho, S.-B. Ho, and C.-H. Tan, “Improving asthma treatment adherence by integrating weather information with responsive web technique,” (Virtual Conference, 2024), p. 030005. https://doi.org/10.1063/5.0216655
35. M.E.K.K. Khairi, S.-B. Ho, and C.-H. Tan, “Real-time tracking and category filtering of an automated system for trading cards collections,” (Virtual Conference, 2024), p. 060001. https://doi.org/10.1063/5.0216659
36. R.K.J. Yik, S.-B. Ho, and C.-H. Tan, “Improving the prediction resolution time for mobile Eczema support system,” (Virtual Conference, 2024), p. 030007. https://doi.org/10.1063/5.0216658
37. M.T.-T. Yong, S.-B. Ho, and C.-H. Tan, “Migraine Generative Artificial Intelligence based on Mobile Personalized Healthcare,” Journal of Informatics and Web Engineering  **4**(1), 275–291 (2025). https://doi.org/10.33093/jiwe.2025.4.1.20
38. S.B. Ho, S.L. Chean, I. Chai, and C.H. Tan, “Engineering Meaningful Computing Education: Programming Learning Experience Model,” in *2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, (IEEE, Macao, China, 2019), pp. 925–929. https://doi.org/10.1109/IEEM44572.2019.8978920
39. S.B. Ho, I. Chai, and C.H. Tan, “AN EMPIRICAL INVESTIGATION OF METHODS, FOR TEACHING DESIGN PATTERNS WITHIN, OBJECT-ORIENTED FRAMEWORKS,” Int. J. Info. Tech. Dec. Mak. **06**(04), 701–722 (2007). https://doi.org/10.1142/S021962200700271X