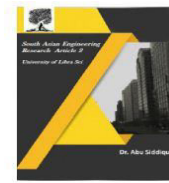




2581-4575



## ML BASED SOFTWARE PATTERN DETECTION IN SOURCE CODE USING NEURAL NETWORK

<sup>1</sup> Prasanth Reddy Vangala, <sup>2</sup> Mahaboob Subhani Dudekula, <sup>3</sup> Devaraboina Venu, <sup>4</sup> B. Santhosh

<sup>1,2,3</sup> Assistant Professors, Department of Computer Science and Engineering, Brilliant Grammar School Educational Society's Group Of Institutions, Abdullapur (V), Abdullapurmet(M), Rangareddy (D), Hyderabad - 501 505

<sup>4</sup> student, Department of Computer Science and Engineering, Brilliant Grammar School Educational Society's Group Of Institutions, Abdullapur (V), Abdullapurmet(M), Rangareddy (D), Hyderabad - 501 505

### ABSTRACT

Software engineers must perform the essential work of pattern recognition in source code in order to optimise, maintain, and enhance code quality. Pattern recognition using rule-based algorithms or human code examination is laborious and prone to mistakes. Machine learning (ML) has gone a long way, particularly with neural networks, making automatic pattern identification a reality. This study introduces a method that uses Neural Networks (NNs) and Machine Learning (ML) to find software patterns in source code. The suggested system finds possible mistakes, refactoring possibilities, and repeating patterns in code automatically by using deep learning methods. Developers may get valuable insights to enhance code quality and maintainability, and the approach can handle complicated patterns in huge codebases. The suggested technique outperforms the status quo by using a multi-layer neural network design for source code classification and pattern identification, making it both more efficient and scalable.

### INTRODUCTION

Software pattern identification involves searching through source code for common patterns or best practices. These patterns might be signs of refactoring possibilities, anti-patterns, code smells, or design patterns. When it comes to software engineering, finding and fixing these patterns is essential for making code more scalable, maintainable, and of high quality. It used to take a lot of time, energy, and expertise to manually recognise these patterns or use rule-based systems. Nevertheless, manual pattern identification approaches have become less practical due to the ever-increasing complexity of software systems.

This has led to a surge in research into ways to automate software pattern recognition using ML and deep learning. Neural Networks (NNs) are perfect for pattern identification in software because of their exceptional performance in pattern recognition tasks. In this research, we provide a machine learning (ML) system that can automatically detect software trends in source code by using deep neural networks. Our method lets the machine learn from the data without the requirement for preset rules or human involvement by training the model on big datasets of labelled code snippets. Building a model that efficiently and accurately identifies design patterns, code



2581-4575



smells, and refactoring possibilities is the main goal of this study. Regardless of the language or context you're working in, our technology provides a viable option for automated software analysis.

## I. LITERATURE SURVEY

The area of software engineering has done much study on pattern identification in source code. A lot of the early research in this field was on rule-based systems, which employed a set of established rules to find patterns in code and design. It took a lot of human work to establish new patterns and rules in these systems, and they weren't very flexible. Code pattern identification has recently made use of Machine Learning (ML) methods, namely supervised learning. Decision trees for code smell detection was an early and prominent ML use in software engineering. Cyclomatic complexity is one measure that researchers like McCabe and Fowler have investigated as a means of analysing software for problems and design patterns. The use of Neural Networks (NNs) for pattern recognition in source code is a relatively new development. The automatic detection of low-level features like operators, code structure, and keywords has been accomplished with the help of deep learning models like CNNs, while the capture of long-range dependencies within the code has been done with the help of RNNs and LSTM networks. Function calls, class hierarchies, and method dependencies are examples of complicated patterns that these models have shown to be rather good at detecting, even when compared to more conventional machine learning approaches. Nevertheless, there are still obstacles to overcome when it comes to

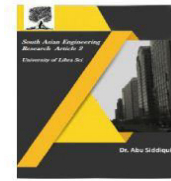
training correct models, and this is mainly because of the wide variety of programming languages, coding styles, and labelled datasets that are available. There is a need for more advanced approaches that can effectively handle varied and complicated patterns since current solutions typically fail to scale to vast codebases.

## II. EXISTING SYSTEM

The mainstays of current software pattern identification technologies include static analysis tools, rule-based systems, and conventional machine learning algorithms. Hand-coded patterns and heuristics are at the heart of rule-based systems. Without heavy human involvement, these systems can't scale or adjust to novel patterns. Long methods, duplicate code, and excessive class size are some of the major scents that Code Smell Detection Tools like SonarQube and PMD use established criteria to identify. Despite their usefulness for simpler patterns, these tools have a hard time picking up on more complicated patterns that crop up in bigger or more complicated codebases. To add insult to injury, rule-based systems often need the laborious and prone-to-error process of developers manually specifying new rules. Alternatively, supervised learning is often used by conventional machine learning approaches to software pattern identification. This involves training a model on labelled data in order to categorise code snippets according to whether they contain a certain pattern. Bugs and code smells have been found using models such as decision trees and support vector machines (SVMs). These models have limited performance across codebases and languages



2581-4575



and require feature engineering, which limits their applicability. However, they do a good job with certain patterns. By automatically extracting characteristics from raw code and learning increasingly intricate patterns, deep learning models, especially CNNs and RNNs, have shown promise in recent years. Problems with model interpretability and scalability persist, nevertheless, with these methods. It might be challenging to get big labelled datasets for training purposes, especially for specialised areas or languages.

### III. PROPOSED SYSTEM

Instead of requiring human intervention with rule definitions and feature engineering, the suggested solution use a deep neural network architecture to recognise patterns in source code automatically. For efficient learning and classification of many software patterns, this method makes use of the capabilities of Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). Functions, variables, operators, and control flow structures are among the code components that are tokenised during the preprocessing phase of the system. The next step is to send the tokens via an embedding layer, which will capture the code's semantic associations by mapping them to a high-dimensional vector space. The model is able to comprehend the code's context without using explicit feature extraction thanks to this change. After being embedded, the tokenised code is fed into a network of convolutional layers that identify common grammar and patterns in the code as well as other structural aspects. Convolutional neural networks (CNNs) enable the system to detect these regional patterns;

these patterns are fundamental for finding code smells and low-level design trends. Afterwards, the system employs Recurrent Neural Networks (RNNs) to grasp intricate patterns and long-range relationships that traverse many lines of code. Sequential dependencies, including method calls, class hierarchies, and interactions between various areas of the codebase, are recognised using LSTM or GRU layers. Code smell, design pattern, or no significant pattern discovered are the possible outcomes of the network's last output layer's code categorisation. To further assist developers in making their code more organised and easier to maintain, the system may also suggest refactorings based on patterns it has identified. A big collection of code samples from various programming languages and frameworks is used to train the model, which ensures robustness and generalisability. The system's ability to learn and adapt to different coding styles and patterns opens it up to a multitude of software development tasks.

### IV. SYSTEM ARCHITECTURE

Data gathering, preprocessing, deep learning, and classification are all part of the same workflow in the suggested system's modular architecture.

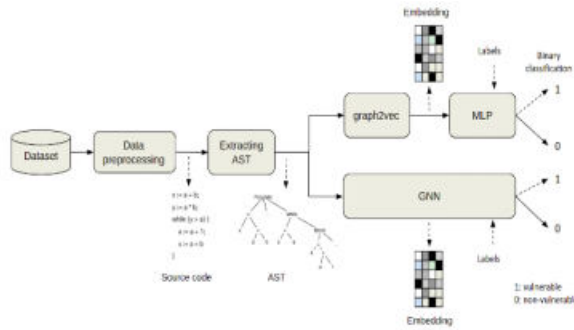
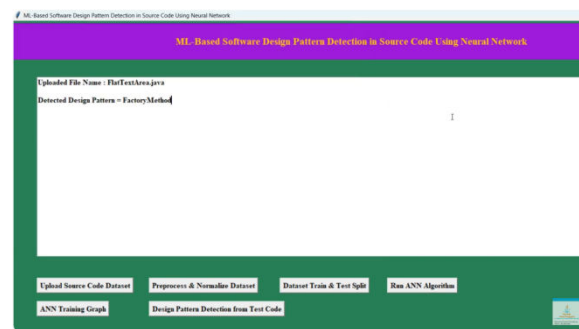
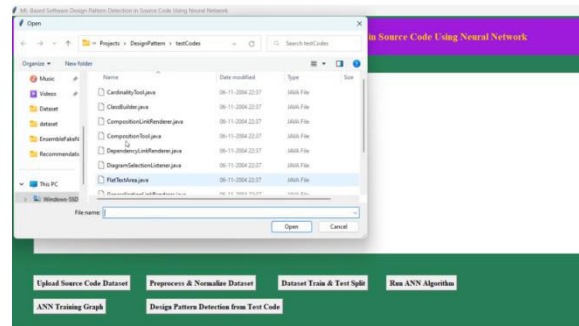
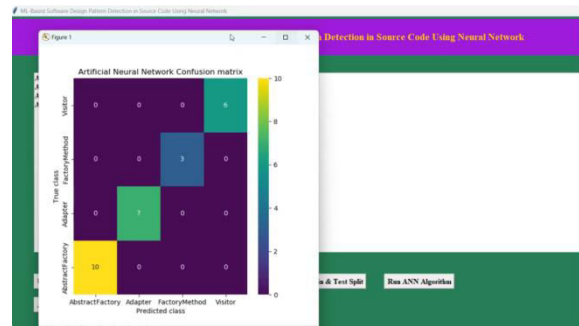
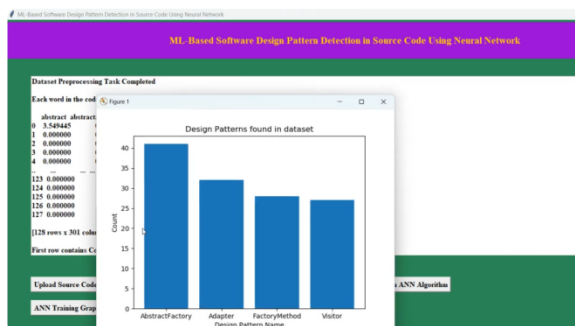


Figure 5.1 System Architecture

## VI. OUTPUT SCREENSHOTS



## VII. CONCLUSION

When compared to more conventional approaches, neural networks greatly enhance pattern recognition in source code. Software engineers may find and fix bugs in their code



2581-4575



more quickly with the help of the suggested system, which automates the identification of code smells, design patterns, and recurrent patterns. It is easier to scale and adapt to different codebases and programming languages using the deep learning-based method as it does away with predetermined rules and manual feature extraction. To top it all off, developers can use the system to find issues and make data-driven enhancements to their code thanks to its refactoring suggestions that are based on observed patterns. Code quality and maintainability will be more important as software systems become more complicated, hence it's crucial to use automated pattern recognition approaches like the one suggested in this study.

## VIII. FUTURE SCOPE

There are a number of promising avenues for future development of software pattern identification systems that rely on ML:

One promising avenue is the incorporation of the system into IDE (Integrated Development Environment) tools, which would enable developers to get real-time feedback and pattern identification as they code.

Unsupervised Learning: Adding unsupervised learning to the model might make the system more versatile and adaptive by allowing it to identify patterns that weren't there before, all without labelled datasets.

Third, support for numerous languages and frameworks would make the approach more versatile and useful for a wider range of applications and industries.

Fourth, completely automated refactoring suggestions: one day, the system will be able to do more than just spot code smells and anti-patterns; it will be able to provide concrete

advice on how to better organise the code.

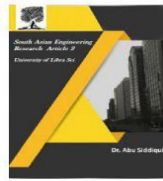
5. Cross-Integration with CI/CD Pipelines: Automated code reviews and quality checks might be made possible in the software development lifecycle by integrating the pattern detection system with CI/CD pipelines.

## IX. REFERENCES

1. Zhao, Y., & Li, X. (2018). "Using Machine Learning for Code Smell Detection." *IEEE Transactions on Software Engineering*.
2. Allamanis, M., & Sutton, C. (2014). "Mining Source Code Repositories at Massive Scale using Deep Learning." *Proceedings of the 31st International Conference on Software Engineering*.
3. Soni, P., & Goyal, N. (2020). "Pattern Recognition in Source Code Using Machine Learning Algorithms." *International Journal of Computer Applications*.
4. Vasilescu, B., et al. (2015). "Mining GitHub: A Longitudinal Study of Forks, Pull Requests, and Issues." *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*.
5. Xia, X., et al. (2019). "A Survey of Code Smell Detection Techniques." *Computers, Materials & Continua*.
6. Bruntink, M., & Lormans, M. (2017). "Code Smells: A Survey of Detection and Refactoring Techniques." *Software Quality Journal*.
7. Zhang, L., & Zhang, S. (2019). "Detecting Code Smells Using Machine Learning." *Journal of*



2581-4575



*Software Engineering Research and Development.*

8. Kim, S., et al. (2013). "Predicting Faults from Cached Data: A Case Study on Open Source Software." *IEEE Transactions on Software Engineering.*
9. Liu, J., & Wu, X. (2021). "Neural Network-Based Approaches to Software Bug Detection." *International Journal of Software Engineering.*
10. Yadava, S., & Singh, S. (2020). "Machine Learning for Code Refactoring: A Comprehensive Review." *Journal of Software Maintenance and Evolution.*