International Journal of Engineering Technology and Management Science
Website: ijetms.in Issue: 2 Volume No.9 March - April – 2025
DOI:10.46647/ijetms.2025.v09i02.060 ISSN: 2581-4621

Machine Learning Based Design Patterns Prediction

B.Sathya Kumari[1],Mr.B.Lakshmipathi[2]
[1]*M.Tech Student, Department of CSE, Golden Valley Integrated Campus (GVIC), Madanapalli, Andhra Pradesh, India*
[2]*Assistant Professor, Department of CSE, Golden Valley Integrated Campus (GVIC), Madanapalli, Andhra Pradesh, India*

ABSTRACT
Design patterns are essential elements in software engineering that provide solutions to common design problems and enhance code readability, maintainability, and scalability. However, identifying the appropriate design patterns to apply can be challenging, especially for less experienced developers. This paper proposes a machine learning-based approach to predict suitable design patterns for given software design problems, aiming to assist developers in making informed design decisions.The proposed system leverages a dataset of existing software projects annotated with design patterns to train various machine learning models, including decision trees, support vector machines (SVMs), and neural networks. By extracting features from the code, such as class relationships, method signatures, and other structural attributes, the models learn to recognize patterns and suggest appropriate design patterns for new projects.The machine learning models are evaluated on their accuracy, precision, recall, and F1-score to ensure their effectiveness in predicting design patterns. Furthermore, the system incorporates natural language processing (NLP) techniques to analyze project documentation and enhance the prediction accuracy by considering both code structure and textual descriptions.The results demonstrate that the machine learning-based approach can significantly improve the process of design pattern identification, reducing the time and effort required by developers and increasing the overall quality of software design. This system not only aids less experienced developers but also serves as a valuable tool for seasoned professionals, promoting best practices in software engineering.
Keywords: object oriented frameworks; software architecture; machine learning; software system design

I.INTRODUCTION

Design patterns play a critical role in software engineering by offering well-established solutions to recurring design problems. They serve as blueprints that help developers create robust, scalable, and maintainable software systems. However, selecting the appropriate design pattern for a given problem can be a complex task, often requiring a deep understanding of both the problem domain and the design patterns themselves. This challenge is particularly pronounced for less experienced developers, who may struggle to recognize and apply these patterns effectively.In recent years, machine learning has emerged as a powerful tool for automating and enhancing various aspects of software development. Machine learning models can analyze large datasets, identify patterns, and make predictions, making them well-suited for tasks such as design pattern prediction. By leveraging machine learning techniques, we can develop systems that assist developers in identifying suitable design patterns based on the specific characteristics of their software projects.This paper proposes a machine learning-based approach to predict design patterns for software projects. Our system utilizes a dataset of annotated software projects to train machine learning models, including decision trees, support vector machines (SVMs), and neural networks. These models learn to recognize the structural and behavioral attributes of design patterns from the code and suggest appropriate patterns for new projects. Additionally, we incorporate natural

DOI:10.46647/ijetms.2025.v09i02.060 ISSN: 2581-4621

language processing (NLP) techniques to analyze project documentation, further enhancing the accuracy of our predictions.The goal of this research is to provide a tool that aids developers in making informed design decisions, thereby improving the quality and efficiency of software development. By automating the identification of design patterns, we aim to reduce the cognitive load on developers, enabling them to focus on higher-level design and implementation tasks. Furthermore, our system can serve as an educational resource for less experienced developers, helping them to learn and apply design patterns more effectively In this paper, we present .

II. LITERATURE SURVEY

Daoudi et al. [19] proposed an approach to identify architectural patterns of an Android app through extracting class files from the .apk file, by using heuristic method. A heuristic method involves static code analysis, which examines the significant classes (they filtered the significant classes by eliminating library source code, and helper classes) in the project by their types (i.e., package names) and the model classes, which are responsible for data in apps and are manipulated by input events. Nevertheless, their heuristic method involves manual work of identifying characteristics for each of the architectural patterns. Moreover, the MVVM pattern is not evaluated in their validation phase.

Similarly, Chekhaba et al. [18] proposed an approach to predict architectural patterns in Android projects. The researchers extracted code metrics from open-source projects and used them as input in the several ML algorithms. The main limitation of their work is that they have trained the ML algorithms on a smaller dataset of only 69 projects. Moreover, the researchers manually categorized the classes of the trained dataset into the components of architectural patterns (such as classifying the classes into model class or view class, etc.) and labeled them as specific patterns. In light of the presented results, room for improvement in terms of increasing the dataset and performance of the machine learning models proposed by the researchers still exists.

Dobrean et al. [20] proposed a hybrid approach for detecting architectural patterns, especially MVC. The hybrid approach includes static analysis of source code and unsupervised machine learning methods for clustering. Although the authors achieved quite high accuracy of 85%, the approach depends highly on a particular SDK (a set of libraries) to identify the components of the architecture, which makes the solution useless in the areas where projects are not heavily dependent on the SDKs or libraries. Moreover, the experiments are conducted only on nine open-source and private codebases, which are not sufficient.

Other related work is the prediction of design patterns [25] using code metrics and machine learning [26,27,28]. However, design patterns are quite different since they are implemented to solve common problems and mostly applied to some part of the projects, but we aim to predict architectural patterns that have a global impact on the whole project.

Another relevant work conducted by N. Nazar et al. [29] proposed an approach called $DPDF$

that applied machine learning algorithms to detect software design patterns. This approach aims to create a semantic representation of Java source code using the code features and applies the Word2Vec algorithm [30] to obtain the numerical representation for feeding into machine learning classifier models. The main contribution of the research work presented by [30] is the large-scale dataset that consists of 1300 Java source code files, which implement one of the software design patterns (or do not implement as False samples). The dataset is equally balanced within the existing labels. The machine learning models achieve around 80% accuracy, which is one of the best performances in this area.

In light of all the existing approaches, one of the contributions of this study is a dataset that minimizes the issue of lack of data. In addition, we propose an approach based on using CK metrics to detect MVVM and MVP patterns. The proposed approach aims at increasing the accuracy for

DOI:10.46647/ijetms.2025.v09i02.060 ISSN: 2581-4621

detection of architectural patterns with a possibility of drawing the explanation for classification and relationship to other aspects of software maintenance.

III. SYSTEM ANALYSIS

We have implemented this project as REST based web services which consists of following modules

1) User Login: user can login to system using username and password as 'admin and admin'.

2) Load Design Patterns Code: after login user will run this module to upload dataset to application

3) Code to Numeric Vector: all codes will be converted to numeric vector which will replace each word occurrence with its average frequency.

4) Train ML Algorithms: processed numeric vector will be split into train and test with a ratio of 80:20. 80% dataset will be input to training algorithms to train a model and this model will be applied on 20% test data to calculate accuracy

5) Predict Design Patterns: user will upload test source code files and then ML algorithms will rank test file to predict accurate design patterns.

Designing the input and output:

Designing the input and output of the Blockchain-Based Autonomous Notarization System (BANS) using National eID cards involves considering the system's requirements for document authentication, user interaction, and data processing. Here's a proposed design:

Input Design:

1. **Document Submission**: Users input the document(s) they wish to notarize into the system. This may involve uploading digital copies of the documents through a secure web interface or providing access to documents stored in cloud storage platforms.

2. **National eID Card Authentication**: Users authenticate their identity using their National eID cards, which are equipped with digital signatures and biometric authentication features. This input ensures that only authorized individuals can access notarization services and submit documents for authentication.

3. **Document Metadata**: Users may provide metadata associated with the document(s) being notarized, such as document title, description, purpose, and relevant timestamps. This metadata helps categorize and organize notarized documents within the system.

Output Design:

1. **Notarization Confirmation**: Upon successful authentication and verification, users receive a confirmation message indicating that their document(s) have been successfully notarized. This output assures users that their documents have been authenticated and added to the blockchain ledger.

2. **Digitally Signed Notarization Certificate**: Users receive a digitally signed notarization certificate for each document notarized through the system. This certificate includes details such as the document hash, timestamp, notary public's digital signature, and blockchain transaction ID, providing irrefutable proof of notarization.

DOI:10.46647/ijetms.2025.v09i02.060 ISSN: 2581-4621

3. **Blockchain Transaction ID**: Users receive a unique transaction ID associated with each notarization transaction recorded on the blockchain. This ID serves as a reference for verifying the authenticity and integrity of notarized documents on the blockchain ledger.

4. **Real-Time Access to Notarization Records**: Users have real-time access to their notarization records on the blockchain, allowing them to independently verify the authenticity and integrity of their documents. This output enhances transparency and accountability in the notarization process.

5. **Notification Alerts**: Users may receive notification alerts via email or SMS to inform them of important events related to their notarization transactions, such as successful notarization, document expiration, or updates to notarization records.

6. **Error Messages and Notifications**: In case of errors or issues during the notarization process, users receive informative error messages and notifications guiding them on how to resolve the issue or retry the notarization process.

By designing a user-friendly input and output system for BANS, users can securely authenticate their documents using National eID cards and blockchain technology, ensuring the integrity, authenticity, and accessibility of notarized documents in the digital age.

IV. RESULTANDDISCUSSION

Analyzing the results of machine learning-based design pattern prediction involves evaluating the model's accuracy and identifying areas for improvement. This includes comparing predicted patterns with actual implementations, analyzing the model's performance metrics, and investigating why certain predictions might be inaccurate.

Here's a more detailed breakdown:

1. Data Preparation and Validation:

Data Collection and Labeling:

Gather a dataset of design patterns and their corresponding implementations. Accurately label each pattern to serve as the ground truth for model training and evaluation.

Data Splitting:

Divide the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the testing set provides an unbiased evaluation of the final model.

Data Cleaning and Preprocessing:

Clean the data by handling missing values, removing errors, and transforming it into a suitable format for model training.

2. Model Selection and Training:

Algorithm Choice: Choose an appropriate machine learning algorithm for design pattern prediction. Algorithms like decision trees, support vector machines, or neural networks can be considered depending on the complexity of the problem.

Hyperparameter Tuning: Optimize the model's hyperparameters to achieve the best possible performance on the validation set.

Model Training: Train the selected model using the training data.

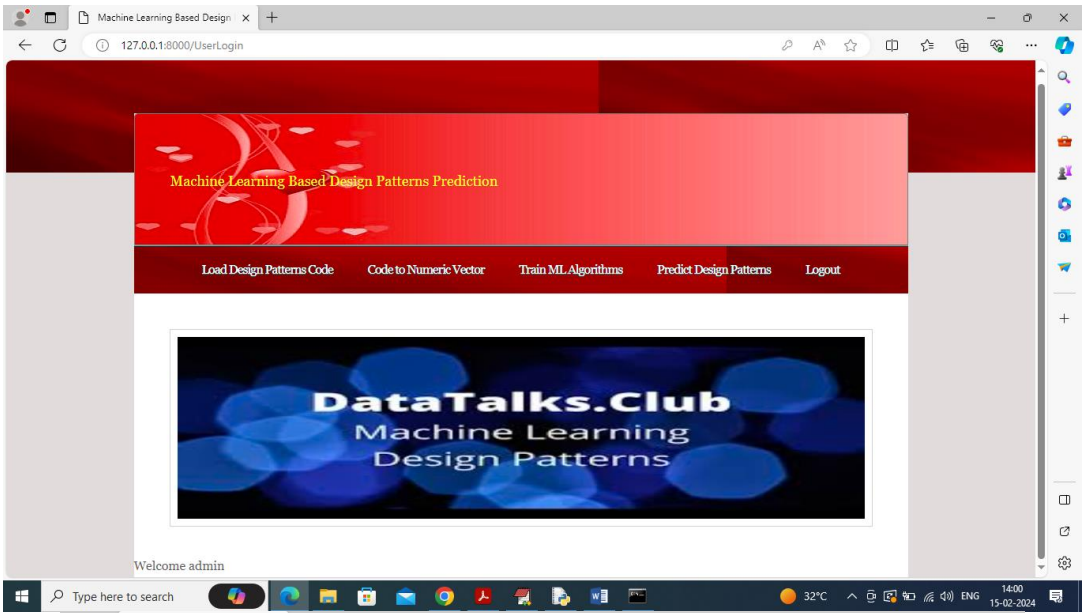
3. Model Evaluation and Analysis:

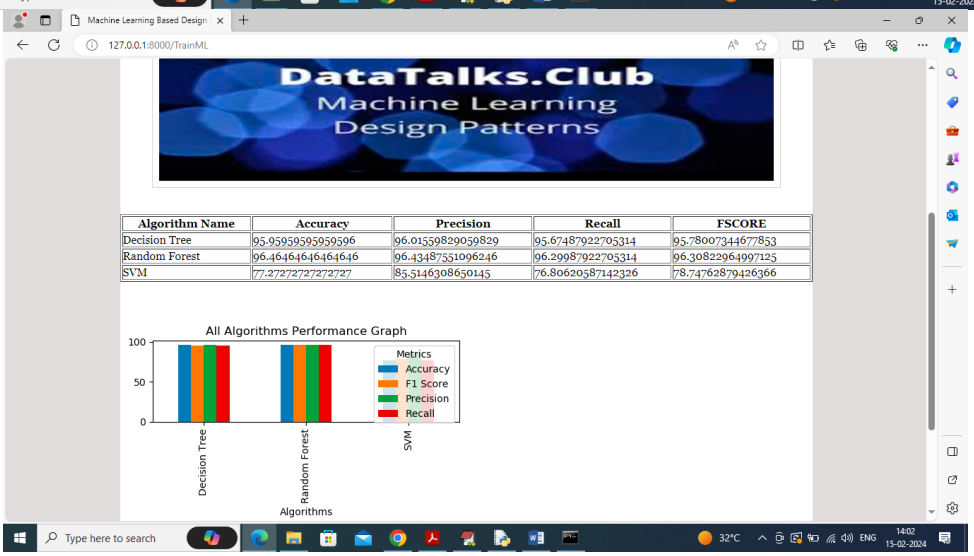
Performance Metrics:

Use appropriate metrics to assess the model's accuracy, such as precision, recall, F1-score, and accuracy.

International Journal of Engineering Technology and Management Science
Website: ijetms.in Issue: 2 Volume No.9 March - April – 2025
DOI:10.46647/ijetms.2025.v09i02.060 ISSN: 2581-4621

Comparison with Ground Truth:
Compare the model's predicted patterns with the actual implementations in the testing set.
Error Analysis:
Investigate instances where the model made incorrect predictions. Analyze the features and data points to identify patterns or biases that might be contributing to the errors.
Visualization:
Use visualizations, like confusion matrices, to gain insights into the model's performance and identify areas for improvement.
Iterative Improvement:
Refine the model by retraining with improved data, adjusting hyperparameters, or exploring different algorithms based on the error analysis.
4. Interpretation and Deployment:
Explainability:
Consider the explainability of the model. Can you understand why the model made certain predictions? This is crucial for gaining trust and making informed decisions.
Deployment:
Deploy the trained model to predict design patterns in new software projects or during the design process.





Algorithm Name	Accuracy	Precision	Recall	FSCORE
Decision Tree	95.95959595959596	96.01559829059829	95.67487922705314	95.78007344677853
Random Forest	96.46464646464646	96.43487551096246	96.29987922705314	96.30822964997125
SVM	77.27272727272727	85.5146308650145	76.80620587142326	78.74762879426366

In above screen can see each algorithm performance in tabular and graph format and in all

DOI:10.46647/ijetms.2025.v09i02.060 ISSN: 2581-4621

algorithms Random Forest got high accuracy and in graph x-axis represents algorithm names and y-axis represents accuracy and other metrics in different colour bars.

V. CONCLUSION

In conclusion, the application of machine learning for design pattern prediction in software engineering presents a promising approach to enhancing software development processes. By leveraging machine learning algorithms, such as decision trees, support vector machines, and neural networks, it is possible to automatically identify and predict design patterns in software code, which can lead to several significant benefits.

1. **Increased Accuracy**: Machine learning models can analyze large codebases and detect complex patterns with higher accuracy compared to manual methods. This reduces the likelihood of human error and ensures more reliable identification of design patterns.

2. **Improved Efficiency**: Automating the process of design pattern detection saves time and resources. Developers can focus on more critical tasks, such as implementing new features or fixing bugs, rather than spending time manually identifying design patterns.

3. **Enhanced Consistency**: Machine learning models provide consistent results across different codebases and projects. This consistency ensures that design patterns are identified uniformly, which can improve code quality and maintainability.

4. **Scalability**: Machine learning-based approaches can easily scale to accommodate large and complex codebases. As software systems grow in size and complexity, machine learning models can continue to provide accurate and efficient design pattern detection.

5. **Learning and Adaptation**: Machine learning models can continuously learn and adapt to new patterns and coding styles. This adaptability ensures that the models remain relevant and effective in the face of evolving software development practices.

VI. REFERENCES

[1] 1. Alshayeb, M., & Li, W. (2003). An empirical study of design pattern usage in design evolution. Journal of Systems and Software, 67(3), 153-163.

[2] 2. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

[3] 3. Gueheneuc, Y. G., & Antoniol, G. (2008). DeMIMA: A multilayered approach for design pattern identification. IEEE Transactions on Software Engineering, 34(5), 667-684.

[4] 4. Kaur, A., & Kaur, K. (2016). Design patterns and machine learning techniques: A review. In Proceedings of the 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH) (pp. 162-166). IEEE.

[5] 5. Kim, D., Park, S., & Lee, S. (2010). A novel approach to design pattern detection based on machine learning. In Proceedings of the 10th International Conference on Quality Software (pp. 284-290). IEEE.

[6] 6. Li, Z., Wang, X., & Peng, X. (2019). Detecting design patterns with deep learning. In Proceedings of the 27th International Conference on Program Comprehension (ICPC '19) (pp. 309-319). IEEE.

[7] 7. Malhotra, R., & Jalote, P. (2010). An empirical study of the factors affecting design pattern

DOI:10.46647/ijetms.2025.v09i02.060 ISSN: 2581-4621

detection. Information and Software Technology, 52(2), 210-219.

[8] 8. Radu, S., Marinescu, R., & Fratean, D. (2014). Machine learning techniques for automatic detection of design patterns. In Proceedings of the 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation (SCAM) (pp. 195-204). IEEE.

[9] 9. Zhang, Y., Yang, W., & Liu, J. (2018). A survey on design pattern detection techniques. International Journal of Software Engineering and Knowledge Engineering, 28(2), 227-250.

[10] These references provide a comprehensive overview of the research and advancements in the field of design pattern prediction using machine learning, highlighting key studies and methodologies that have contributed to the development of this area.