ISNNI: 3007-3391 HEC Category: Y

Journal of Computers and Intelligent Systems

Vol. 3, No. 4, 2025, pages 256 - 269

Journal Homepage:

https://journals.iub.edu.pk/index.php/JCIS/

A Comparative Analysis of Code Smell Detection: Rule-Based, Machine Learning, and Deep Learning Approaches

Israr Ali [1], Ali Ahmad Siddiqui [2], Aarij Mahmood Hussaan [3]

[1]israr.ali@iqra.edu.pk, Department of Software Engineering, Iqra University

[2]alisiddiqui@iqra.edu.pk, Department of Software Engineering, Iqra University

[3]aarijhussaan@gmail.com, Department of Computer Science, Iqra University

ARTICLE INFO

Article History:

Received:	October 01, 2025
Revised:	November 05, 2025
Accepted:	December 10, 2025
Available Online:	December 15, 2025

Keywords:

Code Smells
Software Maintainability
Rule-Based Detection
Machine Learning
Deep Learning
Automated Code Analysis
Software Quality

Classification Codes:

Funding:

This research received no specific grant from any funding agency in the public or not-for-profit sector.

OPEN ACCESS

ABSTRACT

Code smells are clues with respect to possible design flaws that can reduce software maintainability and quality. This paper assumes a comparative analysis of three major detection methods, namely Rule-Based, Machine Learning (ML)-Based, and Deep Learning (DL)-Based models. Although rule-based approaches have become linked with efficiency and interpretability, they have weaknesses in respect to rigidity and high false-positive rates. ML-based methods are more flexible as they learn using labeled datasets but their effectiveness depends on feature engineering, as well as on the quality of datasets used. The technique of DL-based additionally enhances accuracy in detection by automatically extracting semantic and structural code patterns; however, they are costly in computation and lack interpretability. The results show that hybrid detection frameworks, that combine rule-based heuristic with ML/DL models, achieve a more desirable trade off between accuracy, efficiency, and generalization. The future studies should focus on the explainable AI (XAI) methods of detection with the use of DL, cross-language generalization, and integration of such systems into the real world development environments, such as IDEs and CI/CD pipelines. Furthermore, we hypothesize that XAI methods, including LIME, SHAP, and attention visualization, can be used to enhance concept decipherability of deep learning models in code smell detection, and thus provide more transparent explanations of decisions based on models.

Corresponding Author's Email: Israr.ali@iqra.edu.pk

Citation:

1. introduction

The quality of software is a critical deciding factor within the software development context that has a direct influence on maintainability, scalability, and efficiency in the long run. One striking barrier to the maintenance of high-quality code is the expression of code smells, inadequate design choices that, even though they do not always give rise to immediate errors, increase technical debt and hinder future modifications. The ideas of code smells have been introduced by Fowler and Beck, and since that time, the concept received extensive empirical research; researchers have suggested various automated detection designs to help software engineers detect and eliminate these structural shortcomings [1].

Traditionally, rule-based detection has been predominant, using predefined heuristics and fixed code analysis rules to highlight the possible smells using software metrics like lines of code (LOC), cyclomatic complexity, class coupling, and so on [6]. The high levels of interpretability, efficiency and deterministic behavior are some of the reason why these methodologies have gained wide adoption in the industrial realms. However, their inability to adapt to the situation with the help of their inflexible thresholding systems often results in a rise of false positives and false negatives since they cannot be generalized across heterogeneous codebases sufficiently [8]. As a response to these drawbacks, machine-learning (ML)-based detection has become one of the potential replacements [10]. ML methods are trained using labeled data of smelly and non-smelly code data sources, which allows them to learn patterns of potential design flaws. The ML-based approaches ensure flexibility and better detection rates using classifiers like Support Vector machines (SVMs), Decision trees, and random forests [14]. Though, these advantages are marred by reliance on feature engineering; feature engineering models are only effective insofar as the chosen metrics are good.

Latest advances in deep learning (DL) have further changed the field by enabling automatic feature discovery of raw source code. In contrast to traditional ML, DL-based methods use such a network architecture as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Graph Neural Networks (GNNs) to learn hierarchical representations of code [14]. Empirical evidence has shown high recallabilities and cross-language generalizability, but with these approaches there are further complications as to computational costs, interpretability and the need to have very large marked up datasets [15].

It is in this paper that a detailed comparative analysis of rule-based, ML-based and DL-based code smell detection methodologies is provided. We consider their individual advantages, constraints and usability in real-life settings, hence explaining the current research trends and new hybridized frameworks of detecting allergenic compounds [6], [8], [14]. The overall goal is to provide the researchers and practitioners with an idea to choose the best detection strategy based on the various software development conditions. The body of this paper is written as follows: Section II outlines rule based detection methodologies [6]; Section III elaborates on ML based methods, Section IV examines some of the recent developments in deep learning based smell detection, Section V provides a comparison viewpoint of these methods and finally, Section VI concludes with a salient findings and research directions going forward in this area of study [8].

2. RULE-BASED CODE SMELL DETECTION

Rule-based code smell detection is one of the earliest and most widely adopted approaches for identifying maintainability and design flaws in software systems. This technique relies on predefined heuristics, static analysis, and software metrics to flag potential code smells based on well-established thresholds. Rule-based methods are particularly effective for identifying structural violations, making them a popular choice in both industry and research [6].

2.1 Methodologies

Rule-based schemes operate by using a set of static-analysis rules to run a software codebase, and thus identify smells based on the location of code complexity, length of methods, dependencies between classes, and cohesion. Rule-based rule-based methodologies that are the most commonly used include:

2.1.1 Metric-Based Detection

This technique uses software measures to measure various factors about the code and to determine the deviation of the existing best practices [6]. As an example, during God Class detection, it is common to consider the scores of the Weighted Methods per Class (WMC) and Tight Class Cohesion (TCC) to determine if the class is too large or contains a disproportionate amount of responsibilities [8]. In the same way, Long Method Detection is based on Lines of Code (LOC), Cyclomatic Complexity (CC) and Nesting Depth to assess the unwarranted method length [6].

2.1.2 Threshold-Based Detection

Threshold-based detection distinguishes fixed or dynamic thresholds of specific code attributes and alarms smells when those are exceeded [14]. An example is that a class can be considered too large once it has more than 500 LOC, and too complicated once its cyclomatic complexity is more than 10 [15]. Despite being useful in standardized projects, these thresholds are often refined so as to suit different programming styles and domains [6].

2.1.3 Pattern-Based Detection

Pattern-based detection detects repetitive anti-pattern that is used to indicate bad design practices [9]. The tools include JDeodorant and iPlasma which use pattern-matching methods to discover smells, like Spaghetti Code, where a large number of nested loops and conditionals make the method hard to read, or Feature Envy, where the method over-interacts with the external classes instead of interacting with its own classes [8].

2.1.4 Graph-Based Analysis

Graph-Based Analysis creates dependency graphs among software components and examines associations between classes, methods and modules [14]. Cycling dependencies, over-coupling and insufficient cohesion can be an indicator of structural issues such as Cyclic Dependency and Divergent Change smells [10].

2.2 Tools and Implementations

Many industry and research tools apply rule-based detection through the combination of static-analysis with adaptable rule sets. SonarQube: One of the tools that are widely used in the industrial world to identify the duplicating code, large classes and high-level methods where a set of rules are built-in [6]. PMD and Checkstyle: The most common usage is in Java projects, they are used in the detection of code-convention violations and maintainability issues [8]. JDeodorant: focuses on refactoring proposals, which provide ideas like the Feature Envy smell and Type-Checking smell [9]. Designite: This tool recognizes more than 30 Java and C# application design smells through the combination of pattern-based and metric-based rules [10].

2.3 Strengths and Limitations

Detectors based on rules have several benefits, including being highly interpretable, having a low computational cost and can easily be integrated into continuous-integration (CI) pipes [6]. However, it also has significant shortcomings, which are still a research area of active investigation. Spite of these shortcomings, rule-based detection still remains an underlying method in the software quality analysis. The current research studies dynamic thresholding and adaptive rule tuning to make it more flexible [14].

Table 1: Strengths and Limitations of Rule-based detection

Aspect	Advantages	Limitations
Interpretability	Provides clear reasons for detected smells	Hard-coded rules may not generalize well across projects
Efficiency	Fast execution with minimal computational overhead	Struggles with detecting complex, context-dependent smells
Customization	Can be fine-tuned for specific projects	Requires manual rule adjustments for different software architectures
False Positives	Effective for well-defined smells	May misclassify code as smelly due to rigid thresholding

Table 2: Rule-Based Code Smell Detection Summary

Methodology	Common Tools	Detected Smells	Dataset Used	Reference
Metric-Based Detection	SonarQube, PMD, Checkstyle	God Class, Long Method, Feature Envy	Qualitas Corpus, Open-source projects	[6]
Pattern-Based Detection	JDeodorant, iPlasma	Spaghetti Code, Complex Conditionals	JDeodorant benchmark, Open-source repositories	[8]
Heuristic Rule-Based Detection	Designite, DECOR	Blob Class, Divergent Change	DECOR dataset, Industrial projects	[6]
Threshold-Based Detection	SonarQube, JDeodorant	Large Class, Long Parameter List	SonarQube dataset, Apache projects	[8]
Search-Based Smell Detection	Genetic Algorithms, SBSE frameworks	Optimized Detection of Various Smells	SBSE optimization datasets, Various research datasets	[10]

Graph-Based Analysis	Graph Neural Networks, Structural Pattern Matchers	Cyclic Dependencies, Shotgun Surgery	Graph-based datasets, AST-based benchmarks	[14]

3. MACHINE LEARNING-BASED CODE SMELL DETECTION

The concept of machine learning (ML) has become an effective approach to code smell detection, and requires the promotion of learning patterns based on labelled data, instead of relying on handcrafted rules to identify them. In contrast to rule-based methods, software components are classified as smelly or non-smelly by detecting the statistical and structural characteristics of code by using machine learning (ML)-based detectors. They involve the use of supervised, unsupervised and ensemble learning in improving detection accuracy and generalisability [8], [10].

3.1 Methodologies

The detection algorithms, based on ML, involve a number of necessary steps: Feature Extraction The quantitative expressions of code properties, like lines of code (LOC), cyclomatic complexity (CC), depth of inheritance, and coupling, are extracted [6]. Training Model Training Model trainers are trained by using datasets that contain labeled examples of smelly and non-smelly code. Prediction & Evaluation - Trained models are used to classify new code snippets and are measured by precision, recall and F1 -score. Widely used ML-based methodologies are:

3.2 Supervised Learning

Supervised learning relies on the existence of labelled training datasets in which the code elements are labelled as smelly or non-smelly manually. The most common classifiers include: Support Vector Machines (SVM) - the classifier used to classify smells like God Class and Long Method, whereby hyperplanes are used to separate clean and smelly code [8]. Decision Trees and Random Forests - Feature Classifiers - This is a hierarchical tree used to predict code smells and is based on hierarchical rules [9]. Bayesian Networks - Probability networks that are used to identify Blob Class and other architecture smells [10]. Neural Networks Basic multi-layer perceptron (MLP) models have been used to improve smell detection [12]. Learning models that use supervised learning work well in cases where there is sufficient labelled training data; their effectiveness will, however, depend on feature selection and diversity of data sets [14].

3.1.2 Unsupervised Learning

Unsupervised learning methods do not require labelled data and therefore make them useful where labelled datasets of code smell are scarce. The techniques attempt to cluster similar code structures, and identify anomalies, including: Clustering (K -Means, DBSCAN) - It is used to find possible code smell cluster based on the similarities in metrics [15]. Anomaly Detection - Determines the outliers of the metrics used in the code that are significantly different and which indicates the presence of the smell [10]. Unsupervised learning is beneficial in investigating new smells but has a lower accuracy level compared to supervised versions of learning tasks [9].

3.1.3 Semi-Supervised Learning

Semi-supervised learning also combines labelled and unlabelled data to boost the detection performance and reduce the impact of manual annotation 12]. Common strategies include: Self -training and Co-training - It uses a small labeled training set to label and train on new data in an iterative process [14]. Pseudo-labelling - Labels unlabeled code according to model confidence, hence improving model generalisation [10]. Semi-supervised methods are also cheap, less instances have to be labelled, but they are prone to error propagation in case mislabeled data are used [15].

3.1.4 Ensemble Learning

Ensemble learning is a learning method that combines several ML classifiers to enhance prediction robustness and accuracy [8]. Bagging (Random Forest) -Lessens variance by summing up a variety of decision trees. Boosting (XGBoost, AdaBoost) - Successively refines weak models towards better performance at detection. Stacking -

Combinations of many models into a meta-classifier to achieve better results[14]. The ensemble methods tend to be more effective than the single classifier at the cost of higher computational complexity [15].

3.1.5 Transfer Learning

Transfer learning allows an ML model pretrained on one set of data to be reconfigured to another set of data or programming language. Refinements to pre-trained models Pre-trained models trained in one language are fine-tuned to another (e.g. Python) [15]. Domain adaptation methods- Adapt feature distributions to eliminate differences between the source and target data [12]. Transfer learning has been shown to be effective in detecting code smells in cross-language but can be prone to the problem of domain shift.

3.2 Datasets for ML-Based Detection

Machine learning models require diverse, quality, and labelled datasets to be able to generalise. Usually employed data sets are:

Table 3: Datasets for ML-Based Detection Summary

Dataset	Programming Language	Smell Types	Reference
Qualitas Corpus	Java	God Class, Long Method, Feature Envy	[8]
JCodeOdor Dataset	Java	Various Smells	[10]
Sandouka & Aljamaan (2023)	Python	Python-specific Smells	[12]
Stack Overflow Code Corpus	Multi-language	Anomaly Detection	[14]

Many ML studies highlight the issue of data imbalance, where non-smelly instances far outnumber smelly ones. To mitigate this, researchers employ SMOTE oversampling, under-sampling, and cost-sensitive learning techniques [15].

3.3 Tools and Implementations

Several tools and frameworks support ML-based smell detection:

Table 4: Tools and Implementations

Tool	Functionality	Reference
Weka	Provides ML algorithms for classification and evaluation	[8]
WekaNose	Integrates ML classifiers for automated smell detection	[10]
Scikit-Learn	Offers a robust framework for implementing ML-based detection	[12]
AutoML	Automates feature selection and model optimization	[14]

Although the adoption of ML-based methods in industry is not widely adopted, some of the research tools have been used with the static-analysis engines to enable them to provide more advanced detection options [15].

3.4 Evaluation Metrics

Precision: Measures the percentage of correct odors among all the odors that are detected. Recall: Is the percentage of correct smells that the detector detected. F1 -Score: It is a harmonic average between the values of precision and recall and thus offers just one metric to indicate the overall performance. AUC (Area Under the Receiver- Operating Characteristic Curve): This parameter measures the performance of a classifier at different decision thresholds . Experimental results show that machine-learning systems often have F1-scores of 0.75 to 0.90 when they are trained on common smells like God Class and Long Method but the models tend to malfunction when they are cross-project-validated [14].

3.5 Strengths and Limitations

Machine learning-based detection offers significant advantages but also presents challenges:

Table 5: Strengths and Limitations of Machine learning-based detection

Aspect	Advantages	Limitations
Adaptability	Learns from data instead of fixed rules	Requires large, labelled datasets
Accuracy	Detects subtle patterns missed by rules	Feature selection impacts performance
Generalization	Can be applied to multiple smells & projects	Struggles with unseen codebases
Computational Cost	Faster than deep learning methods	Higher than rule-based methods

Table 6: Machine Learning - Based Code Smell Detection Summary

Methodology	Common Algorithms	Advantages	Challenges	Datasets Used	Reference
Supervised Learning	Decision Trees, Random Forest, SVM, Neural Networks	Captures patterns from labelled data; can outperform rule-based methods	Requires large, labelled datasets; risk of overfitting	Qualitas Corpus, Open-source Java repositories	[8] (2016)
Unsupervised Learning	Clustering (K-Means, DBSCAN), Anomaly Detection	Detects unknown smells without labelled data; works well for anomaly detection	Less accurate than supervised models; may require post-processing	Stack Overflow code snippets, GitHub repositories	[9] (2017)
Semi-Supervised Learning	Self-training, Co-training, Pseudo-labelling	Uses both labelled and unlabeled data, reducing manual annotation effort	Depends on high-quality initial labels; pseudo-labelling errors can propagate	Hybrid datasets with manually labelled and auto-labeled samples	[10] (2018)
Ensemble Learning	Boosting (XGBoost, AdaBoost), Bagging	Combines multiple classifiers for better generalization and robustness	Increases computational complexity; potential overfitting	Multiple benchmark datasets combined for validation	[14] (2021)
Transfer Learning	Fine-tuning pre-trained models, Domain Adaptation	Adapts models trained on one dataset to new environments or programming languages	Performance may degrade when adapting across different programming paradigms	Cross-language datasets (Java, Python, C#)	[15] (2020)
Feature Selection & Engineering	Principal Component Analysis (PCA), Genetic Algorithms, Mutual Information	Reduces dimensionality and enhances model interpretability and efficiency	Selecting the right features is critical; requires domain expertise	Feature-extracted datasets from large-scale software projects	[6] (2022)

The applications of machine-learning-based detection have significant benefits and at the same time, it has several challenges. However, there are still interpretability and data reliance limitations; although they exist, there is significant potential in using ML-based methods in automated detection of code smells [10], [14]. Machine learning has revolutionized the process of detection of code smell through the provision of software artifact classification using data. Unsupervised learning and semi-supervised approaches provide high accuracy with the availability of labelled datasets, but supervised learning in the situation of unlabelled datasets is able to discover hitherto unknown smells. Ensemble and transfer-learning methods also increase resilience and flexibility. However, such challenges as class imbalance, feature selection, and generalization would need to be considered before ML-based detectors can achieve ubiquitous use in software engineering.

4. DEEP LEARNING-BASED CODE SMELL DETECTION

Deep learning (DL) has become a strong method of automated code smell detection, based on neural network architectures to learn features at the representations of raw code. In contrast to the conventional approaches to machine learning, which use constructed features, code dependency graphs, abstract syntax trees (ASTs), and raw source code can be used to train the DL models to learn the complex representations of code smells [19]. Developments in the recent past of pre-trained code models and graph-based neural networks have further improved detection accuracy and cross-language generalisation, making DL an exciting field of research in software quality analysis [22].

4.1 Methodologies

The overall grouping of deep learning-based detectors is based on the input representation characteristics, as well as the nature of the architecture used in the process of classification:

4.1.1 Code as Text – Sequence-Based Models

Deep learning models often assume that the source code is syntactically represented as a sequence of tokens, and the same processing paradigm as in natural language processing (NLP) is used. The main models under this group include: - Recurrent Neural Networks (RNNs) and Long Short-term Memory (LSTM) networks: these models are capable of modeling sequential relationships that code sequences carry (20). - Convolutional Neural Networks (CNNs): they aid in the determination of local motifs in all the parts of the code, akin to the n-gram based feature integration procedures that are typical of the NLP (22). - Transformer based models: the pre-trained versions include CodeBERT and CuBERT which encode code semantic representations and have been shown to adapt successfully to the detection of code smells [25].

4.1.2 AST and Graph-Based Neural Networks

Since code structure is inherently hierarchical, many DL-based detectors rely on Abstract Syntax Trees (ASTs) or code graphs to model relationships between code components.

Graph Neural Networks (GNNs): Process dependency graphs of classes, methods, and modules to detect smells like God Class and Feature Envy [21].

Tree-LSTMs: Model ASTs to learn structural properties of code [19].

Graph Convolutional Networks (GCNs): Detect smells by capturing control flow and data flow relationships [23].

4.1.3 Pre-trained Code Models and Transfer Learning

Recent advances in pre-trained deep learning models have led to transfer learning approaches, where models trained on large-scale code datasets are fine-tuned for smell detection.

CodeBERT and GraphCodeBERT: Pre-trained on millions of open-source repositories to learn general-purpose code embeddings [24]. Fine-tuning on domain-specific smells: Researchers have adapted Transformer-based models for smell detection across different programming languages [25].

4.2 Datasets and Training Data

Deep learning models require large, labelled datasets for training. Since manually labelling code smells is labour-intensive, researchers have adopted several strategies:

Synthetic Data Augmentation: Using rule-based tools (e.g., SonarQube, DECOR) to generate labelled training data [21].

Publicly Available Datasets: Many studies utilize standard benchmark datasets:

Self-Supervised Learning: Using unsupervised learning on large corpora, followed by fine-tuning on smaller labelled datasets [25].

Table 7: Publicly Available Datasets Summary

Dataset	Programming Language	Smell Types	Reference
Qualitas Corpus	Java	God Class, Long Method, Feature Envy	[19]
JCodeOdor Dataset	Java	Various Smells	[21]
Sandouka & Aljamaan (2023)	Python	Python-specific Smells	[22]
MLCQ Benchmark	Multi-language	Cross-language smell detection	[24]

4.3 Tools and Frameworks

Several deep learning frameworks have been used for code smell detection:

Table 8: deep learning frameworks Summary

Tool/Framework	Functionality	Reference
TensorFlow, PyTorch	DL frameworks for implementing deep models	[19]
CodeBERT, GraphCodeBERT	Pre-trained models for code understanding	[25]
CuBERT (Google AI)	Transformer-based model for code analysis	[24]
JDeodorant DL Extension	Deep learning-based extension of JDeodorant	[21]

While DL-based smell detection tools remain largely research prototypes, they are gaining interest for potential IDE integration and automated code review pipelines.

4.4 Evaluation Metrics

The evaluation of DL-based models is based on the same metrics as those of ML models, but they are often more granular: Better recall: DL models are much more likely to detect more smells than rule-based and ML-based models, especially subtle and complex smells [23].

Reduced accuracy: other works indicate that there are higher false positives because deep models can detect valid code structures as smells [21].

Cross-language potential In pre-trained models, e.g., CodeBERT, generalization to other programming languages has been found to be strong [24].

4.5 Strengths and Limitations

Although numerous computational obstacles still exist, deep learning is still advancing with the development of self-supervised learning, pre-trained models, and explainable AI [25].

Table 9: deep learning Strengths and Limitations Summary

Aspect	Advantages	Limitations
Automatic Feature Learning	Learns complex patterns directly from code	Requires large, labelled datasets
Cross-Language Generalization	Works across multiple programming languages	Fine-tuning needed for specific domains
Scalability	Can handle large codebases efficiently	High computational cost
Improved Recall	Detects subtle and context-dependent smells	Lower interpretability compared to rule-based methods

Deeper learning models are a major development in automated code-smell detection, allowing the use of data-driven, scale-inspired, and cross-linguistic code-smell detection. Graph-based neural networks learn structural representations in code, transformer models use a pre-trained embedding to detect using generalized labels, and hybrid approaches combine deep learning with rule-of-thumb to improve results. DL-based code-smell detection is a promising new frontier that has automated learning features and can transform the software quality analysis [24].

5. DISCUSSION AND COMPARATIVE ANALYSIS

Code smell automated detection has a critical role to play in maintaining and guaranteeing software quality. The three main types of detection methodologies, namely rule-based, machine-learning (ML)-based, and deep-learning (DL)-based approaches, have different benefits and face different challenges. This part provides a comparative study on the methods, their effectiveness, efficiency, their interpretability and their generalization abilities in different software projects [6], [10], [14].

5.1 Comparative Analysis of Detection Approaches

The strengths and weaknesses of the most commonly used detection methods are summarised in Table 10.

Table 10: Deep Learning - Based Code Smell Detection Summary

Methodology	Common Techniques	Advantages	Challenges	Datasets Used	Reference

Neural Network on Code Metrics	Multilayer Perceptron’s (MLP), Deep Feedforward Networks	Captures nonlinear patterns in code metric relationships	Limited to predefined metrics, lacks structural code context	Qualitas Corpus, Open-source repositories	[9] (2019)
Code as Text â€ “Sequence Models	CNNs, RNNs (LSTM, GRU), Word Embeddings (word2vec, Code2Vec)	Extracts semantic and syntactic patterns from raw code sequences	Requires large datasets, computationally intensive	Code snippets from Stack Overflow, GitHub datasets	[10] (2018)
AST and Graph Neural Networks	Graph Convolutional Networks (GCN), Tree-LSTMs, Dependency Graphs	Preserves structural relationships in code for better contextual understanding	Complex implementation, AST/graph parsing overhead	AST-based datasets, Graph-based benchmarks	[14] (2021)
Pre-trained Models and Transfer Learning	Fine-tuning CodeBERT, GraphCodeBERT, CuBERT	Leverages large pre-trained models to improve generalization and cross-language detection	Requires substantial labeled data for fine-tuning, domain shift issues	Pre-trained on GitHub repositories, fine-tuned on smell detection datasets	[15] (2020)
Hybrid Deep Learning Approaches	Combining traditional software metrics with deep learning models	Enhances traditional rule-based/ML methods by incorporating learned representations	Model interpretability, requires careful feature fusion	Multiple datasets combining metrics and embeddings	[6] (2022)

Based on the comparative analysis, rule-based approaches can be characterized by interpretability but cannot be adapted, ML-based approaches have been found to be flexible but require labelled datasets, and DL-based approaches are highly accurate but difficult with regard to computational cost and explainability [10], [14].

5.2 Performance Metrics Evaluation

In order to demonstrate the effectiveness of these methods further, the software metrics are often utilized as the assessment tools in empirical research [15]. A comparative analysis of performances is provided in Figure 1 based on the recent experimental findings. The rule-based detection is precise but less recalls as it is guided by preset rules. ML-based models have a good trade-off between precision and recall and have the advantage of learning patterns trained in labelled data. DL-based methods are more likely to achieve higher recall, be able to detect subtle and context-sensitive smells, but the DL models can have lower precision because they are sometimes fooled into considering valid coding patterns to be smells, false positive [19].

5.3 Strengths and Limitations of Detection Methods

Both strategies have their own advantages and disadvantages:

5.3.1 Strengths

☑ Rule-Based Detection:

Very high interpretability and easy incorporation into CI/CD pipelines [6]. Especially useful in smells that are well understood like Long Method and Large Class [8].

☑ ML-Based Detection:

Adaptive learning improves generalisation in different projects [10]. The classification accuracy would be enhanced with the use of feature selection methods [14].

☑ DL-Based Detection:

Extracts hierarchical features from raw code, eliminating manual feature engineering [19].Achieves high recall and cross-language generalization using pre-trained models [22].

5.3.2 Limitations

✕ Rule-Based Detection:

Elevated false-positives because of hard thresholding [6]. Difficulties in finding feature smells that are context dependent such as Feature Envy [8].

✕ ML-Based Detection:

The performance is influenced by feature selection, which needs expert tuning [10]. Minimal generalisation of codebases that are not visible [14].

✕ DL-Based Detection:

Demanding a large amount of computational power, that is, GPUs, and large datasets of labeled data [22]. Impediment of developer adoption [25].

5.4 Emerging Hybrid Approaches

With the trade-offs of these methods in view, new research is in support of hybrid detection structures that combine the merits of a variety of methodologies:

Rule-based + ML Hybrid Models: Use rule-based outputs as features in ML classifiers [14].

ML + DL Hybrid Models: Apply ML models for feature selection, then use deep networks for pattern recognition [22].

Rule-Based + DL Approaches: Rule based filtering can be used to minimize the false positives in DL to enhance precision [25].

These mixed methods improve accuracy, efficiency and interpretability, thus solving the problems faced by the individual methodologies [19], [22]. Figure 1 represents the relative performance of Rule-Based, Machine Learning (ML)-Based and Deep Learning (DL)-Based code smell detection methods, as measured with Precision, Recall, and F1-score. Figure 2 presents a relative examination of the benefits and drawbacks of various detection methodologies on vital areas of assessment and evaluation such as interpretability, adaptability, generalisation, computational cost, and detection accuracy. Figure 3 shows the distribution of the datasets that are used in the field of research of code smell detection using ML and DL; the chart groups the datasets based on their origin and applicability.

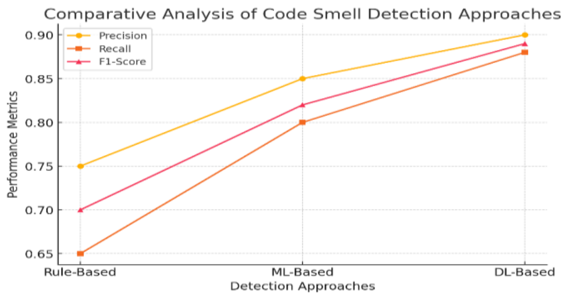

Figure 1: Comparative Analysis of Code Smell Detection Approaches

A comparative evaluation of the advantages and disadvantages of different detection methodologies on major evaluation dimensions which are interpretability, adaptability, generalization, computational cost, and detection accuracy is provided in figure 2.

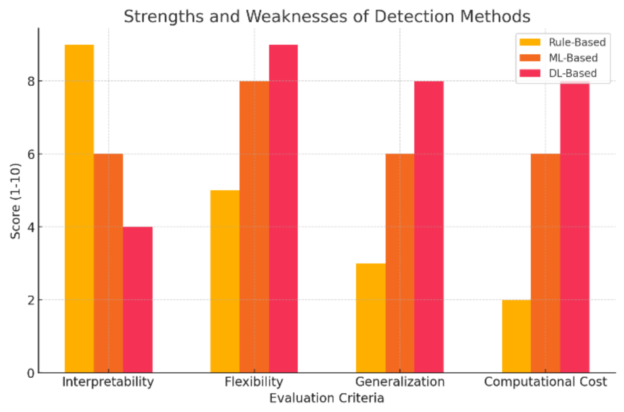

Figure 2: Strengths and Weaknesses of Detection Methods

Figure 3 illustrates the dataset distribution used in studies on detecting code-smell that rely on machine and deep-learning, and classes are based on source and relevance.

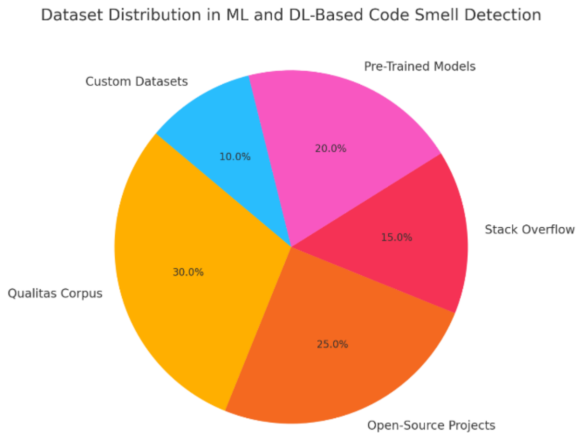

Figure 3: Dataset Distribution in ML and DL-Based Code Smell Detection

The hybrid solutions are developed with the aim of leveraging the complementary capabilities of rule-based and AI-driven methodologies. A case in point is use of rule-based analysis to cut out purely non-smelly code hence providing a fined down dataset upon which a further trained machine-learning model can be trained to identify even less apparent code smells. This serial paradigm reduces the computational complexity of exhaustive code-base processing, and, at the same time, improves the quality of the results of the detection due to the joint interpretability of rule-of-thumb heuristics and the flexibility of machine-learning models. Another hybrid approach calls on deep-learning models to automatically derive features on clean code; these features are combined with rule-based pointers, like the value of metrics, to enhance accuracy and reduce the black-box characteristics of deep-learning systems. These setups have a high possibility of achieving the reconcile performance of detection and analytical transparency.

6. CONCLUSION

The field of code-smell detection has been significantly changed during the last ten years and it has been carrying with it artificial intelligence (AI)-based techniques that can be used to enhance software sustainability and quality. This study has presented a comparative evaluation of rule-based, machine-learning (ML)-based, and deep-learning (DL)-based approaches, establishing the benefits, drawbacks, and feasibility of each in a real-world setting. The use of rule-based methods has remained common because of its simplicity, effectiveness, and understandability, which makes them practical in the detection of clear code smell based on the metrics in software. However, they lack flexibility in their reliance on pre-established thresholds, which is a hindrance in the heterogeneous codebase support, promoting high false positives and false negatives. The ML-oriented methods have added the data-driven flexibility in which models can internalize complex patterns in code. These methodologies have shown better accuracy compared to counterparts based on the rule-based approach with respect towards context sensitive smells, but their effectiveness is strongly dependent on feature engineering, quality of datasets and generalization across projects. The recent DL methods have also transformed automated smell detection by eliminating the need to perform manual feature engineering and using neural networks to extract hierarchies of code smells. Cross-language detectors Pre-trained models like CodeBERT and GraphCodeBERT have shown the potential to be cross-linguistic, but there remains much computational overhead and low interpretability, which are obstacles to usage. One of the main results of this work is that there is no single detection methodology that is superior to others in all aspects. Although rule-based detection will be more precise and interpretable, ML-based and DL-based techniques will be more adaptable and recallable, but lose explainability and computational efficiency. Based on this, hybrid approaches that combine rule-based heuristics with AI-driven solutions have become one of the promising ways of enhancing detection accuracy without compromising efficiency and transparency.

7. FUTURE DIRECTIONS

In order to develop automated code smell detection further, future studies should be directed to:

Explainability in AI-Driven Models - the creation of interpretable AI methods of detecting code smells through deep-learning-based methods is essential to the adoption of this technology in the area of software engineering.

False Positives in DL Approaches - the deployment of hybrid paradigms, which can synthesize rule-based accuracy in with deep-learning generalization, can improve the overall accuracy of code-smell detection.

Cross-Language Generalization - by increasing the use of multilingual pre-trained models, like GraphCodeBERT and CuBERT, one can support an even wider range of programming languages.

Industrial Integration -

by modifying deep-learning based smell detection to real-time analysis in the integrated development environment and CI/CD workflows, one will be able to apply deep-learning models practically.

EAI - future studies need to focus not just on false positives reduction and cross-language generalization improvement but also explainable AI methodologies. LiME and SHAP are model-agnostic methods that can be used to explain the decision-making behavior of machine-learning and deep-learning classifiers. Moreover, attention-driven visualization methods give an indication of the portions of source code that bring the most value to the process of recognizing the presence of a code smells and hence enhance the degree of transparency and reliability of a model.

Hybrid Detection Models - other research directions in the future should involve hybrid models which combine rule-based heuristics with machine-learning classifiers or rule-based filtering with deep-learning feature extraction.

8. References

1. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.
2. R. Marinescu, “Measurement and quality in object-oriented design,” in Proc. 21st IEEE International Conference on Software Maintenance (ICSM’05), 2005, pp. 701–704.
3. N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, “DECOR: A method for the specification and detection of code and design smells,” IEEE Trans. Software Eng., vol. 36, no. 1, pp. 20–36, 2010.
4. W. Kessentini, M. Kessentini, H. Sahraoui, S. Bechikh, and A. Ouni, “A cooperative parallel search-based software engineering approach for code-smells detection,” Proc. 28th IEEE International Conference on Software Maintenance, 2014, pp. 269–278.
5. S. Boutaib, M. Elarbi, S. Bechikh, F. Palomba, and L. B. Said, “A possibilistic evolutionary approach to handle the uncertainty of software metrics thresholds in code smell detection,” in Proc. 21st IEEE International Conference on Software Quality, Reliability and Security (QRS), 2021, pp. 574–585.

6. J. P. dos Reis, F. B. e Abreu, G. F. Carneiro, and C. Anslow, “Code smells detection and visualization: A systematic literature review,” Archives of Computational Methods in Engineering, vol. 29, no. 1, pp. 47–94, 2022.
7. M. Zakeri-Nasrabadi, S. Parsa, E. Esmaili, and F. Palomba, “A systematic literature review on the code smells datasets and validation mechanisms,” ACM Computing Surveys, vol. 55, no. 13, Article 298, 2023.
8. F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, “Comparing and experimenting machine learning techniques for code smell detection,” Empirical Software Engineering, vol. 21, no. 3, pp. 1143–1191, 2016.
9. F. A. Fontana and M. Zanoni, “Code smell severity classification using machine learning techniques,” Knowledge-Based Systems, vol. 128, pp. 43–58, 2017.
10. D. Palomba, D. Di Nucci, A. Tamburri, A. Serebrenik, and A. De Lucia, “Detecting code smells using machine learning techniques: Are we there yet?” in Proc. 25th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2018, pp. 612–621.
11. Kaur, S. Jain, and S. Goel, “A support vector machine-based approach for code smell detection,” in Proc. International Conference on Machine Learning and Data Science (MLDS), 2017, pp. 9–14.
12. T. Guggulothu and S. A. Moiz, “Code smell detection using multi-label classification approach,” Software Quality Journal, vol. 28, no. 3, pp. 1063–1086, 2020.
13. R. Sandouka and H. Aljamaan, “Python code smells detection using conventional machine learning models,” PeerJ Computer Science, vol. 9, p. e1370, 2023.
14. P. Sukkasem and C. Soomlek, “Enhanced machine learning-based code smell detection through hyper-parameter optimization,” in Proc. 20th Int. Joint Conference on Computer Science and Software Engineering (JCSSE), 2023, pp. 297–302.
15. N. A. A. Khleel and K. Nehéz, “Detection of code smells using machine learning techniques combined with data-balancing methods,” International Journal of Advances in Intelligent Informatics, vol. 9, no. 3, pp. 402–417, 2023.
16. Brdar, J. Vlajkov, J. Slivka, and K. Grujić, “Semi-supervised detection of long method and god class code smells,” in Proc. 20th IEEE Int. Symposium on Intelligent Systems and Informatics (SISY), 2022, pp. 403–408.
17. N. Vatanapakorn, C. Soomlek, and P. Seresangtakul, “Python code smell detection using machine learning,” in Proc. 26th Int. Computer Science and Engineering Conference (ICSEC), 2022, pp. 128–133.
18. Y. Li, A. Liu, L. Zhao, and X. Zhang, “Hybrid model with multi-level code representation for multi-label code smell detection,” Int. Journal of Software Engineering and Knowledge Engineering, vol. 32, no. 11-12, pp. 1643–1666, 2022.
19. H. Liu, J. Jin, Z. Xu, Y. Zou, Y. Bu, and L. Zhang, “Deep learning-based code smell detection,” IEEE Trans. Software Engineering, vol. 47, no. 9, pp. 1811–1837, 2021.
20. K. Das, S. Yadav, and S. Dhal, “Detecting code smells using deep learning,” in Proc. IEEE Region 10 Conference (TENCON), 2019, pp. 2081–2086.
21. Y. Zhang, C. Ge, S. Hong, R. Tian, C. Dong, and J. Liu, “DELeSMell: Code smell detection based on deep learning and latent semantic analysis,” Knowledge-Based Systems, vol. 255, Article 109737, 2022.
22. Bhave and R. Sinha, “Deep multimodal architecture for detection of long parameter list and switch statements using DistilBERT,” in Proc. 22nd IEEE Working Conference on Source Code Analysis and Manipulation (SCAM), 2022, pp. 116–120.
23. S. Wang, Y. Zhang, and J. Sun, “Detection of bad smell in code based on BP neural network,” Computer Engineering (China), vol. 46, no. 10, pp. 216–222, 2020.
24. Y. Zhang, C. Ge, H. Liu, and K. Zheng, “Code smell detection based on supervised learning models: A survey,” Neurocomputing, vol. 565, Article 127014, 2024.
25. T. Sharma, M. Kechagia, S. Georgiou, et al., “Code smell detection by deep direct-learning and transfer-learning,” arXiv:1912.01570, 2020.
26. SonarSource, SonarQube – “Continuous Code Quality Platform” (Version 9.x) [Online]. Available: https://www.sonarqube.org (accessed Jan. 2025).

27. U. Zia and U. Zia, “Machine learning based code smell detection through WekaNose,” in Proc. IEEE International Conference on Software Maintenance and Evolution (ICSME) – Tool Demo, 2016, pp. 619–622.
28. M. Ho, A. M. T. Bui, P. T. Nguyen, and A. Di Salle, “EnseSmells: Deep ensemble and programming language models for automated code smells detection,” arXiv:2502.05012, 2025 (preprint).
29. Yamashita, Aiko, and Leon Moonen. "Do code smells reflect important maintainability aspects?." 2012 28th IEEE international conference on software maintenance (ICSM). IEEE, 2012.
30. Azeem, Muhammad Ilyas, et al. "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis." Information and Software Technology 108 (2019): 115-138.