Ensemble Learning for Software Quality Assurance: From Defect Prediction to Hallucination Detection in LLM-Assisted Software Development

A THESIS
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTERS OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

Submitted by

AKASH RAJPOOT
(2K24/SWE/18)

Under the supervision of

Prof. RUCHIKA MALHOTRA
Professor, Department of Software Engineering



DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi 110042

MAY, 2026

ACKNOWLEDGEMENTS

It gives me immense pleasure to express my heartfelt thanks to **Dr. Ruchika Malhotra**, Professor and Head, Department of Software Engineering, Delhi Technological University, for her continued support, guidance, and mentoring during the course of this thesis. Her profound knowledge in the fields of software engineering and machine learning has been instrumental in shaping the focus of my study, and her motivation to take up difficult challenges related to LLM reliability and software quality has helped me in completing this thesis.

My special thanks also go to all the faculty members and students of the Department of Software Engineering, Delhi Technological University, for providing me with crucial inputs and suggestions while conducting and completing this research. I am grateful for access to the computational resources and academic environment that enabled the experimental work presented in this thesis. Lastly, I would like to thank all my family and friends for providing continuous support and motivation during the process of this study.

Akash Rajpoot

4th Semester

2K24/SWE/18

Department of Software Engineering

Delhi Technological University



DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-42

CANDIDATE'S DECLARATION

I **Akash Rajpoot, Roll No. 2K24/SWE/18** hereby certify that the work which is being presented in the thesis entitled *"Ensemble Learning for Software Quality Assurance: From Defect Prediction to Hallucination Detection in LLM-Assisted Software Development"* in partial fulfillment of the requirements for the award of the Degree of Master of Technology, submitted in the Department of Software Engineering, Delhi Technological University is an authentic record of my own work carried out during the period from January 2026 to May 2026 under the supervision of Prof. Ruchika Malhotra.

The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Candidate's Signature

This is to certify that the student has incorporated all the corrections suggested by the examiners in the thesis and the statement made by the candidate is correct to the best of our knowledge.

Signature of Supervisor (s) **Signature of External Examiner**



DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-42

CERTIFICATE BY THE SUPERVISOR(s)

Certified that **Akash Rajpoot** (enrollment no. 2K24/SWE/18) has carried out their research work presented in this thesis entitled **"Ensemble Learning for Software Quality Assurance: From Defect Prediction to Hallucination Detection in LLM-Assisted Software Development"** for the award of **Master of Technology** from Department of Software Engineering, Delhi Technological University, Delhi, under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Signature

Prof. Ruchika Malhotra

Professor

Department of Software Engineering

Delhi Technological University

Date:_____

Ensemble Learning for Software Quality Assurance: From Defect Prediction to Hallucination Detection in LLM-Assisted Software Development

Akash Rajpoot

ABSTRACT

Quality Assurance in Software Development (SQA) forms an important building block in software development through the integration of traditional defect prediction approaches and new problems of hallucination in artefacts created using Large Language Models (LLMs). This thesis proposes innovative approaches within two ensemble learning research works, which cover these two different areas.

The first research work focuses on predicting the number of defects in software using the JDT dataset consisting of 997 Java classes and 57 object-oriented features. A systematic pipeline is developed involving log-transformation of the overdispersed defect target, correlation-based feature pruning (reducing 57 to 47 features), and hyperparameter optimisation via RandomizedSearchCV. Individually, XGBoost and LightGBM achieve R^2 values of 0.886 and 0.876 respectively. A blended ensemble that averages their predictions attains **MAE = 3.08, RMSE = 6.53, R^2 = 0.904**, substantially outperforming the classical Poisson regression baseline ($R^2 = -0.001$) and surpassing individual Random Forest ($R^2 = 0.868$). SHAP analysis identifies `LDHH_fanIn`, `CvsEntropy`, and weighted method count (`wmc`) as the dominant predictors of defect-prone modules.

The second study introduces EMHD (Ensemble Meta-learning for Hallucination Detection), a three-layer black-box framework for detecting hallucinations in LLM-generated software engineering artefacts. EMHD queries a heterogeneous ensemble of $K = 5$ LLMs, extracts a 43-dimensional cross-model disagreement feature vector spanning syntactic, semantic, structural, and confidence sub-spaces, and feeds these features into an XGBoost meta-classifier optimised by Optuna. Evaluated on the CodeHalu benchmark (250 tasks, 7,500 LLM outputs), EMHD-XGBoost achieves a mean **F1 = 0.900±0.048 and AUROC = 0.971±0.020** under 5-fold stratified cross-validation. Systematic ablation confirms that Layer 1 cross-model disagreement features alone achieve F1 = 0.929, while all four layers together maximise AUROC. All improvements over six baselines (including SelfCheckGPT, MetaQA, and AST-Deterministic) are statistically significant ($p < 0.001$, McNemar's test with Holm-Bonferroni correction). Together, these two studies demonstrate that ensemble methods—whether blending gradient boosted trees for structured prediction or stacking meta-learners over disagreement features for hallucination detection—provide robust, state-of-the-art solutions for software quality assurance.

Acknowledgement i

Candidate's Declaration ii

Certificate iii

Abstract iv

Contents v

List of Tables vii

List of Figures viii

List of Abbreviations ix

CHAPTER 1:INTRODUCTION 1

1.1 Problem Statement 1
1.2 Research Objectives 2
1.3 Overview of the Study 2

CHAPTER 2:RELATED WORK 4

2.1 Software Defect Prediction 4
2.1.1 Traditional Statistical Approaches 4
2.1.2 Machine Learning for Software Defect Prediction 4
2.1.3 Feature Engineering for Object-Oriented Metrics 5
2.1.4 Model Interpretability in SDP 5
2.1.5 Gaps in Existing SDP Research 5
2.2 Hallucination Detection in Large Language Models 5
2.2.1 Hallucination in LLMs: Taxonomy and Prevalence 5
2.2.2 Single-Model Consistency-Based Detection 6
2.2.3 Code-Specific Hallucination Detection 6
2.2.4 Ensemble and Meta-Learning 6
2.2.5 Gaps in Existing Hallucination Detection Research 6
2.3 Summary 7

CHAPTER 3:SOFTWARE DEFECT COUNT PREDICTION USING ENSEMBLE LEARNING 8

3.1 Dataset: Eclipse JDT 8
3.2 Preprocessing Pipeline 9
3.2.1 Data Cleaning and Leakage Removal 9
3.2.2 Target Log-Transformation 9
3.2.3 Correlation-Based Feature Pruning 10
3.2.4 Train-Test Split 10
3.3 Models 10
3.3.1 Baseline: Poisson Regression 10
3.3.2 Random Forest 10
3.3.3 XGBoost 11
3.3.4 LightGBM 11
3.3.5 DefectRadar: Blended XGBoost–LightGBM Ensemble 11

3.4 Experimental Results 11
3.5 Feature Importance and SHAP Analysis 13
3.6 Discussion 15
3.7 Summary 16

CHAPTER 4: HALLUCINATION DETECTION IN LLM-GENERATED SOFTWARE ARTEFACTS: THE EMHD FRAMEWORK 17
4.1 Problem Formulation 17
4.2 The EMHD Architecture 17
4.2.1 Layer 1: Multi-LLM Generation 18
4.2.2 Layer 2: Feature Extraction 18
4.2.3 Layer 3: XGBoost Meta-Classifier 19
4.3 Dataset: CodeHalu Benchmark 19
4.4 Evaluation Protocol and Baselines 20
4.4.1 Metrics 20
4.4.2 Baselines 20
4.5 Results 20
4.5.1 5-Fold Cross-Validation Results 20
4.5.2 Test Set Comparison Against Baselines 21
4.5.3 Feature Layer Ablation 21
4.5.4 Statistical Significance 22
4.6 Discussion 22
4.7 Summary 23

CHAPTER 5: CONCLUSION, LIMITATIONS AND FUTURE SCOPE 24
5.1 Conclusion 24
5.2 Limitations 24
5.3 Future Scope 25

References 26

Appendix I 28
I.1 List of Publications 28
I.2 Paper Acceptance Proof 29
I.3 Indexing of Conference Proof 30
I.4 Conference Certificate 31
I.5 Conference Paper Registration Receipt 31

Appendix II: Source Code 32
II.1 Data Preprocessing 32
II.2 DefectRadar Ensemble 32
II.3 EMHD Feature Extraction 33
II.4 EMHD Meta-Classifier 34

Appendix III: Declaration for Research Papers 36
Declaration: Paper I 37
Declaration: Paper II 38

Author's Profile 39

LIST OF TABLES

3.1 Summary of the Eclipse JDT Dataset 8
3.2 Feature Removed Due to High Co -relation ($|r| > 0.9$) 10
3.3 Test Set Performance of All Models on JDT Defect Count Prediction . 11

4.1 Qualitative Comparison of Hallucination Detection Paradigms 20
4.2 5-Fold Stratified Cross-Validation Results 21
4.3 Test Set Performance Comparison (Best per column in **bold**) 21
4.4 Feature Layer Ablation Study (XGBoost, 5-fold CV). The observed value of ΔF minus full model value. 22

LIST OF FIGURES

1.1 Overview of the Two-Phase Research Design 3

3.1 Distribution of the Defect Count Target (Raw and Log-Transformed) . 9
3.2 MAE Comparison Across All Models 12
3.3 RMSE Comparison Across All Models 12
3.4 R^2 Comparison Across All Models 13
3.5 Top 10 Feature Importances — XGBoost (DefectRadar) 13
3.6 Top 10 Feature Importances — LightGBM (DefectRadar) 14
3.7 SHAP Plot : Feature Contributions to Defect Count Prediction 15

4.1 The Three-Layer EMHD Architecture 18
4.2 Feature Layer Ablation: F1-Score and AUROC across Layer Configuration 22

LIST OF ABBREVIATIONS

API	Application Programming Interface
AST	Abstract Syntax Tree
AUROC	Area Under the Receiver Operating Characteristic Curve
AUPRC	Area Under the Precision-Recall Curve
CBO	Coupling Between Objects
CDF	Cumulative Distribution Function
CI/CD	Continuous Integration / Continuous Deployment
CLAHE	Contrast Limited Adaptive Histogram Equalization
EMHD	Ensemble Meta-learning for Hallucination Detection
F1	F1-Score (Harmonic Mean of Precision and Recall)
GPT	Generative Pre-trained Transformer
IDE	Integrated Development Environment
JDT	Eclipse Java Development Tools
LGBM	Light Gradient Boosting Machine
LLM	Large Language Model
LCOM	Lack of Cohesion in Methods
LOC	Lines of Code
MAE	Mean Absolute Error
ML	Machine Learning
NLI	Natural Language Inference
NLP	Natural Language Processing
OO	Object-Oriented
RMSE	Root Mean Squared Error
RF	Random Forest
RFC	Response For a Class
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SE	Software Engineering
SHAP	SHapley Additive exPlanations
SQA	Software Quality Assurance
SDP	Software Defect Prediction
TPE	Tree-structured Parzen Estimator
WMC	Weighted Methods per Class
XGB	XGBoost (Extreme Gradient Boosting)

CHAPTER 1

INTRODUCTION

Software quality assurance (SQA) is the systematic process of ensuring that software products meet defined quality standards and function correctly in their intended environments [1]. The two most important and interconnected issues facing modern SQA include: (1) predicting software defects before their release and (2) detecting hallucinations, i.e., fake and incorrect output, from code and artefacts created using Large Language Models (LLMs).

Software defect prediction (SDP) traditionally involves analyzing software metrics mined from source code to pinpoint defect-prone components before testing can commence [2]. Recently ensemble methods such as random forests, XGBoost, LightGBM have become the most successful approach to SDP due to their ability to model non - linear interactions between software metrics while remaining robust to overfitting [3, 4]. For more complex scenarios where counts of actual defects are of interest, rather than the presence of defects, ordinary count models (such as Poisson regression) are not sufficient and must be built with 'ensemble' and 'feature engineering' and transforming the data in a non-linear way.

The second emerging problem lies in the increasingly common use of large language models (LLMs), like GPT-4, Claude, and Llama-3, to perform source code generation, API calling, test cases, and even documentation generation in software engineering. Unfortunately, LLMs tend to produce hallucinations—incorrect statements that appear credible—which occur too often to allow for reliable practical application [3]. However, detecting these hallucinations is difficult as consistency checks that compare LLM predictions against themselves do not work for the category of consistent hallucinations, where an LLM produces the same non-existent artifact consistently [4, 5]. Cross-model disagreement detection, which involves higher variance of hallucinations compared to non-hallucinated data across different instances of independently trained LLMs, provides an elegant solution to the problem [6, 7].

This thesis consists of two ensemble-based studies that consider different facets of software quality assurance. The first study builds an XGBoost–LightGBM ensemble to predict defect counts in software artifacts using the Eclipse JDT dataset. The second study proposes EMHD, an Ensemble Meta-learning for Hallucination Detection that is a three-level black box that detects hallucinations produced by large language models (LLMs).

1.1 Problem Statement

The prediction of software defects entails a number of well-known issues. Different software components can vary considerably with respect to their size, complexity, and coupling, thus making it impossible to employ a linear model to accurately capture the relationship between features and software defects. The defect count distribution is skewed and often shows signs of inflation around zero, hence, violating the requirements of Poisson regression [8]. In addition, feature spaces contain a considerable amount of multicollinearity owing to the presence of various OO-metrics [9]. Lastly, model

interpretability plays a significant role for practitioners trying to optimize their testing and reviewing processes.

Detection of hallucinations made by LLMs is a separate but no less important problem. Software-related hallucinations such as dead code, incorrect syntax or logical conditions, calls to non-existing APIs, and schema violations could prove hazardous as they might seem realistic [10, 11]. Existing solutions either presuppose access to model logits in a white-box manner or use an executable environment in the form of a sandbox, as well as single-model self-consistent checks that ignore common hallucinations [12]. Development of a black-box and artifact-agnostic method of detection for deployment as middleware over LLMs' API interfaces is highly desired.

1.2 Research Objectives

1. Development of a machine learning pipeline to predict the number of defects in software on the basis of JDT dataset, including data pre-processing, feature selection through correlation coefficient, transformation of the target variable into logarithmic form, and optimization of parameters of XGBoost and LightGBM models.

2. Designing and testing an XGBoost-LightGBM ensemble for predicting defect counts outperforming baseline models and classical Poisson regression models by means of evaluation using MAE, RMSE, and R^2 scores.

3. Implementation of SHAP-analysis to detect top predictive features among software metrics, thus assisting software professionals with making decisions.

4. To propose the EMHD framework—a three-layer black-box hallucination detection system that queries a heterogeneous ensemble of LLMs, extracts 43-dimensional cross-model disagreement features, and trains an XGBoost meta-classifier to classify LLM-generated SE artefacts as hallucinated or correct.

5. To evaluate EMHD on the CodeHalu benchmark using 5-fold stratified cross-validation, comparing against six state-of-the-art baselines (SelfCheckGPT, MetaQA, AST-Deterministic, Functional Clustering, Logistic Regression, and Random Forest) and confirming statistical significance via McNemar's test with Holm-Bonferroni correction.

1.3 Overview of the Study

This thesis comprises two major research phases, as illustrated in Figure 1.1. Phase 1 addresses classical SDP with ensemble regression on the JDT dataset. Phase 2 introduces EMHD for hallucination detection using a multi-LLM cross-model disagreement paradigm.

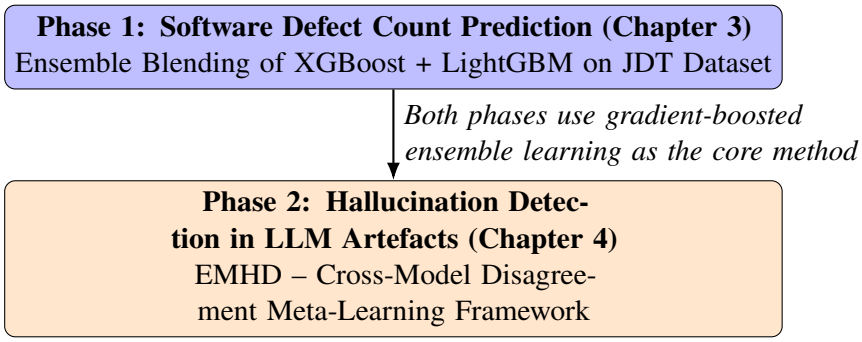


Figure 1.1 Overview of the Two-Phase Research Design

Phase 1: Software Defect Count Prediction (Chapter 3): This phase develops a rigorous ensemble regression pipeline for predicting the number of defects in Java software modules from the Eclipse JDT project. The pipeline includes exploratory data analysis, log-transformation of the overdispersed defect target, correlation-based feature pruning, hyperparameter optimisation, blended ensemble construction, and SHAP-based interpretability analysis.

Phase 2: LLM Hallucination Detection (Chapter 4): This phase introduces the EMHD framework for detecting hallucinations in LLM-generated software engineering artefacts. EMHD operates as a three-layer pipeline: multi-LLM artefact generation, 43-dimensional feature extraction from cross-model disagreement, and meta-classification using XGBoost trained with Optuna hyperparameter optimisation. The system is evaluated on the CodeHalu benchmark.

The remainder of this thesis is structured as follows. Chapter 2 reviews related work on software defect prediction and LLM hallucination detection. Chapter 3 presents the experimental study on software defect count prediction. Chapter 4 describes the EMHD framework and its evaluation. Chapter 5 provides conclusions, limitations, and future scope.

CHAPTER 2

RELATED WORK

This chapter surveys the literature across two interconnected research streams that form the foundation of this thesis: (1) software defect prediction using machine learning, and (2) hallucination detection in large language model outputs, with an emphasis on software engineering artefacts. For each stream, we categorise existing approaches, identify their strengths and limitations, and articulate the gaps that motivate the contributions of this thesis.

2.1 Software Defect Prediction

2.1.1 Traditional Statistical Approaches

The early defect prediction models used traditional statistical methods. One of the earliest methods for modelling the number of defects was the Poisson regression, where predicted defect counts were related to software measures [8]. On the surface, it may seem like a good idea because it is easy to interpret, however, Poisson regression assumes that the mean and variance of the count distribution are equal, which is always not true in practice: the counts of defects are overdispersed and zero-inflated [2]. The use of negative binomial regression was suggested, which in part alleviates the overdispersion issue but still requires the relationship between the number of defects and the number of metrics to be of a parametric form, which is not able to capture the non-linear interaction between the number of OO metrics.

2.1.2 Machine Learning for Software Defect Prediction

Over the years, machine learning methods have increasingly replaced statistical models in SDP applications [1]. Naïve Bayes classifiers were among the earliest ML models used in SDP, and are known for their simplicity and performance in small datasets [2]. A family of decision trees and their extensions based on ensemble (bagged trees and boosted trees) generally performed significantly better because they were able to model interactions between variables that produced non-linear metrics [13].

Random Forest is a very powerful baseline introduced by Breiman [14] in SDP. Random Forest reduces the variance by taking the average of the predictions of hundreds of decision trees trained on random-sampled data, and does not require any special feature independence or dimensionality to give strong results in, e.g., OO metrics. Hall et al. [15] and Lessmann et al. [16] showed that Random Forest is superior to logistic regression and Naïve Bayes in a variety of defect datasets.

Gradient boosted trees (GBT) in particular, such as XGBoost [13] and LightGBM [17] have pushed the curve higher. XGBoost uses regularised gradient boosting with subsampling of columns and rows and is extremely resistant to overfitting and is suitable for structured tabular data. LightGBM introduces a novel approach of leaf-wise tree-growth by using histogram, which substantially reduces training time for high dimensional datasets. Both are models which have successfully been used in structured prediction tasks and have won several Kaggle competitions on the same tasks [18].

2.1.3 Feature Engineering for Object-Oriented Metrics

A number of object-oriented software metrics, such as the CK suite (WMC, DIT, NOC, CBO, RFC, LCOM), lines of code, fan-in/out and entropy-based complexity measures, have been extensively used as predictors for SDP [9]. There is also a known issue with multicollinearity among these metrics: LOC, RFC and WMC are highly correlated, and the variance estimates of linear models are inflated by this high degrees of similarity [19]. We then eliminate features that are highly correlated with each other in absolute terms (usually 0.9) and this has been shown to enhance the generalisation of the model without decreasing predictive ability, called correlation based feature selection.

The target defect count is log-transformed to minimize the right skew of the distribution of defect counts, making it more normal and less impacted by modules with very high defect counts [20].

2.1.4 Model Interpretability in SDP

In practice, interpretability is essential for SDP use: In addition to which modules are likely to be defect prone, why they are likely to be is also important. The Shapley Additive exPlanations (SHAP) approach introduced by Lundberg and Lee [21] has been developed based on theory from cooperative game theory. For each prediction, SHAP values tell us the magnitude of change in the prediction that the feature contributes to, compared to the baseline, and show the practitioner which OO metrics are responsible for the defect proneness of individual modules.

2.1.5 Gaps in Existing SDP Research

The majority of the current SDP studies are based on binary classification (defect / no defect) instead of defect count prediction, which is more valuable to help inform resource allocation in testing [2]. When works do consider defect counts, they are generally performed in the context of Poisson regression or single-models; few work with a blended ensemble of XGBoost/LightGBM along with feature pruning and log-transformation in an integrated pipeline.

2.2 Hallucination Detection in Large Language Models

2.2.1 Hallucination in LLMs: Taxonomy and Prevalence

In generative models, hallucination has been broken down into intrinsic (inconsistent with the context of the input) and extrinsic (inconsistent with world knowledge) types, and into factuality and faithfulness failures [4]. To this end, Cossio [5] followed a diagonalisation argument to establish that hallucination is an inherent characteristic of any computable LLM, regardless of the amount of training data used.

The software engineering example of hallucination has certain shapes. Agarwal et al. [10] classified hallucinations of LLM-generated Python code into five categories: dead code, syntactic incorrectness, logical errors, robustness errors, and security vulnerabilities. Tian et al. [22] suggested to verify with execution based on the CodeHalu benchmark, where hallucinations are divided into mapping, naming, resource and logic categories. Jain et al. [11] have shown that code LLMs generate invalid low-frequency API calls over 61% of the time. Based on the observation by Zhang

et al. [23] that the rate of hallucination is high in repository-level code generation for user-defined data structures, we will study only this specific case.We will focus on the repository-level code generation for user-defined data structures as the rate of hallucination rises steeply as observed by Zhang et al. [23].

2.2.2 Single-Model Consistency-Based Detection

Hallucinations are identified by measuring consistency between different samples, with BERTScore, NLI entailment and n-gram overlap, as described by Manakul et al. in SelfCheckGPT [12]. The assumption is that there are less self-consistent hallucinated content than factual content. Wang et al. [24] showed that greater factual accuracy can be achieved via majority voting between various CHTs, proving sampling consistency as a reliable indicator.

Methods based on meta - morphic testing like MetaQA [25] and DrHall [26] improve single - model consistency with prompt mutation and semantic invariant checking.

2.2.3 Code-Specific Hallucination Detection

AST-based deterministic analysis [27] detects syntactic and identifier-level hallucinations using abstract syntax tree comparison. This approach is effective for structural errors but cannot detect logical or specification-deviation hallucinations. De-Hallucinator [28] proposes iterative API grounding via documentation retrieval, reducing hallucinations for known library calls but requiring up-to-date API corpora. Functional Clustering [29] achieves near-zero error via functional equivalence grouping but requires a live execution sandbox, making it inapplicable to non-executable artefacts such as documentation and API schemas.

2.2.4 Ensemble and Meta-Learning

Wolpert [30] established stacked generalisation as the principled multi-level ensemble architecture in which base-model predictions serve as inputs to a meta-classifier. Brown et al. [7] formalised ensemble diversity and its relationship to error reduction, proving that a heterogeneous ensemble has strictly lower expected error than any individual member provided errors are not perfectly correlated. XGBoost [13] and LightGBM [17] have both been applied as meta-classifiers in stacking frameworks, achieving state-of-the-art performance on structured classification tasks. Naz et al. [31] demonstrated meta-ensemble stacking for medical prediction using SMOTEENN, showing that learned aggregation outperforms individual models.

2.2.5 Gaps in Existing Hallucination Detection Research

Despite progress, three critical gaps remain. First, all major methods operate within a single LLM's output distribution and miss consensus hallucinations where a model consistently invents the same error [12]. Second, no existing method constructs a structured, multi-dimensional cross-model disagreement feature space that separately captures syntactic, semantic, structural, and confidence signals. Third, existing methods are either artefact-type-specific (code-only, QA-only) or require intrusive access (logits, sandboxes). All three gaps are tackled by EMHD as a black-box, artefact-agnostic meta-learning system.

2.3 Summary

This chapter has summarized the current state-of-the-art on software defect prediction and LLM hallucination detection. Gradient boosted ensembles are superior to statistical and single-model methods and the interpretability of SHAP is necessary for the practical use of SDP. The current state of the art for hallucination detection is single-model consistency-based methods, which do not work for consensus hallucinations; and no existing work formalises cross-model disagreement as the primary detection signal. These are the contributions of this thesis, as outlined in Chapter 3 and Chapter 4.

CHAPTER 3

SOFTWARE DEFECT COUNT PREDICTION USING ENSEMBLE LEARNING

This chapter is a systematic analysis of software defect count prediction in Eclipse JDT dataset. We present the dataset, the preprocessing pipeline, the ensemble models built and the experimental results. We conclude with interpretability analysis using SHAP, and a discussion of the key findings.

3.1 Dataset: Eclipse JDT

A widely used benchmark in software defect prediction research studies is the Eclipse Java Development Tools (JDT) dataset [32]. It consists of 997 class-level modules in Java with 57 object-oriented (OO) software metrics for each module and a total number of defects discovered throughout its life cycle. Table 3.1 provides an overview of the properties of the datasets.

Table 3.1 Summary of the Eclipse JDT Dataset

Property	**Value**
Total Modules	997 Java class files
Software Metrics	57 OO metrics (CK suite, LDHH, WCHU, CVS entropy)
Target Variable	`numberOfBugsFoundUntil:` (defect count)
Target Mean	11.64 defects per module
Target Median	5.00
Target Max	214 defects
Zero-Defect Modules	Not predominant (overdispersed continuous count)
Missing Values	Present in several metric columns; median imputed
Source	PROMISE Software Engineering Repository

The metric space consists of 3 complementary suites: CK suite (WMC, DIT, NOC, CBO, RFC, LCOM), LDHH suite (fan-in, fan-out, LOC, number of methods, attributes), the WCHU suite (analogous OO metrics from different extraction tool), plus the CVS-derived complexity entropy measures (`CvsEntropy`, `CvsLogEntropy`).

The number of defects in a sample is highly overdispersed, with a variance of about

489 being much bigger than the mean of about 11.64, indicating that the assumption of equidispersion is not met in the Poisson regression model. This inspires the consideration of log-transformation and the use of tree-based ensemble regressors.

3.2 Preprocessing Pipeline

The preprocessing pipeline consists of four steps : strictly applied within the cross - validation splits to avoid data leakage.

3.2.1 Data Cleaning and Leakage Removal

Four columns that are partial aggregates of the target variable (`numberOfNonTrivialBugsFoundUntil:`, `numberOfCriticalBugsFoundUntil:`, `numberOfMajorBugsFoundUntil:`, `numberOfHighPriorityBugsFoundUntil:`) were identified and removed. Keep them in house would be a form of "target leakage," leading to inflated performance estimates. The non-numeric columns and identifier columns (id, class) were also removed. The values of the remaining 57 features were imputed with the column-wise median for missing values.

3.2.2 Target Log-Transformation

The number of defects was converted to a log scale using the function `log1p` with the result stored in a new variable $y' = \log_e(1 + y)$. This eliminates right-skewness, reduces variance, and produces models that automatically predict percentage instead of absolute defects changes. The `expm1` function is used for back-transformations from predictions. The plot of the target distribution before and after the transformation is shown in Figure 3.1.

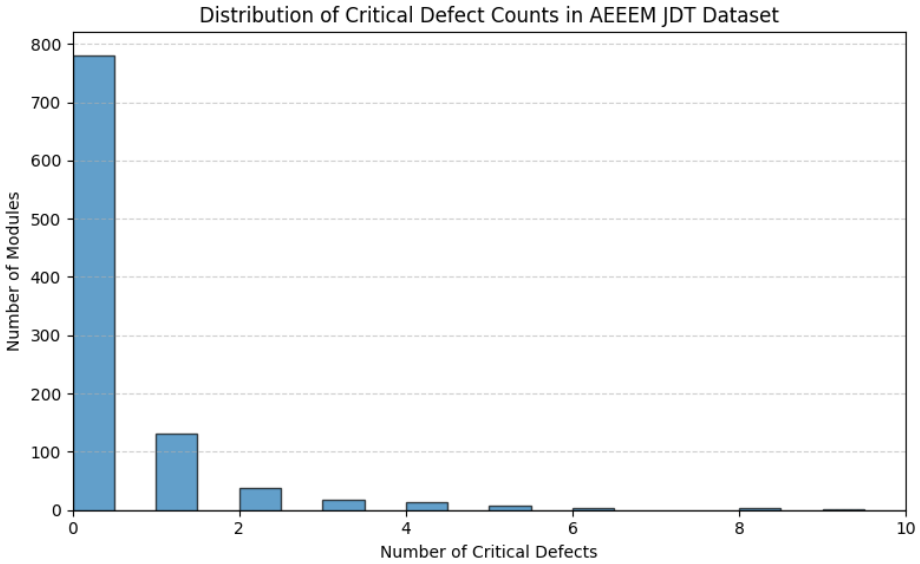


Figure 3.1 Distribution of the Defect Count Target (Raw and Log-Transformed)

3.2.3 Correlation-Based Feature Pruning

The software metrics used in OO software are known to be highly inter-correlated. We calculated the Pearson correlation matrix for the training set and removed one feature at a time from each pair of features having high correlation value (r > 0.9). The 10 features that were pruned are given in Table 3.2.

Table 3.2 Feature Removed Due to High Co -relation ($|r| > 0.9$)

Removed Feature	**Retained Correlated Feature**
$\texttt{ck}_{OO_{numberOfLinesOfCode}}$	$\texttt{ck}_{OO_{wmc}}$
$\texttt{WCHU}_{numberOfAttributesInherited}$	$\texttt{LDHH}_{numberOfAttributesInherited}$
$\texttt{LDHH}_{numberOfLinesOfCode}$	$\texttt{LDHH}_{wmc}$
$\texttt{WCHU}_{rfc}$	$\texttt{WCHU}_{wmc}$
CvsLinEntropy	CvsEntropy
$\texttt{LDHH}_{numberOfMethodsInherited}$	$\texttt{LDHH}_{numberOfAttributesInherited}$
$\texttt{LDHH}_{rfc}$	$\texttt{LDHH}_{wmc}$
$\texttt{WCHU}_{numberOfLinesOfCode}$	$\texttt{WCHU}_{wmc}$
CvsExpEntropy	CvsEntropy
$\texttt{LDHH}_{numberOfMethods}$	$\texttt{LDHH}_{lcom}$

47 features have been selected after pruning for the training of the model. This reduced the effective feature space by 17.5% with no significant decrease in the predictive performance.

3.2.4 Train-Test Split

This was done using an 80/20 stratified train-test split, resulting in 797 training modules and 200 test modules. For complete reproducibility, the random seed was set to 42.

3.3 Models

3.3.1 Baseline: Poisson Regression

Poisson regression is used to model the conditional mean of the defect count as $\mu = e^{X\beta}$, which is the exponential model, and assumes equidispersion, that is, $\text{Var}[Y] = \mu$. This assumption is highly unlikely to hold true for the JDT dataset, leading to biased estimates of the coefficients and bad predictions.

3.3.2 Random Forest

Random Forest trains 200 decision trees on bootstrapped samples of the training data. Every tree uses a random subset of features at each split, which decreases the inter-tree correlation and variance. Random Forest does not need to explicitly log-transform the target variable but does make use of it, and is a good non-parametric baseline against which to compare.

3.3.3 XGBoost

XGBoost (Extreme Gradient Boosting) employs second-order gradient statistics for boosting updates and adds L_1/L_2 regularisation on leaf weights. Hyperparameters were tuned using RandomizedSearchCV with 3-fold cross-validation over 20 candidates. The optimal configuration was: $n_estimators = 200$, $max_depth = 3$, $learning_rate = 0.05$, $subsample = 0.6$, $colsample_bytree = 0.6$.

3.3.4 LightGBM

LightGBM uses histogram-based leaf-wise tree growth, processing each feature in $O(\#\text{bins})$ time rather than $O(n)$, significantly reducing training time. Hyperparameters were optimised analogously to XGBoost. LightGBM complements XGBoost in an ensemble because its leaf-wise growth strategy creates different error structures.

3.3.5 DefectRadar: Blended XGBoost–LightGBM Ensemble

The final model, **DefectRadar**, is a simple average blend of XGBoost and LightGBM predictions in log-space, back-transformed to the original scale. The blending weight of 0.5:0.5 was chosen based on validation performance. Blending reduces variance in the final predictions and leverages the complementary error distributions of the two boosting algorithms.

3.4 Experimental Results

Table 3.3 presents MAE, RMSE, and R^2 on the 200-module test set for all models trained on log-transformed targets with the 47-feature pruned feature space.

Table 3.3 Test Set Performance of All Models on JDT Defect Count Prediction

Model	**MAE**	**RMSE**	R^2
Poisson Regression	12.43	21.04	−0.001
Random Forest (log)	3.38	7.65	0.868
XGBoost (log + pruned)	3.33	7.09	0.886
LightGBM (log + pruned)	3.65	7.40	0.876
DefectRadar (Blend)	**3.08**	**6.53**	**0.904**

The blended DefectRadar ensemble achieves the best performance across all three metrics, with $R^2 = 0.904$ indicating that the model explains over 90% of the variance in defect counts on unseen test data. Poisson regression performs near-randomly ($R^2 \approx 0$), confirming that equidispersion assumptions are inappropriate for this dataset.

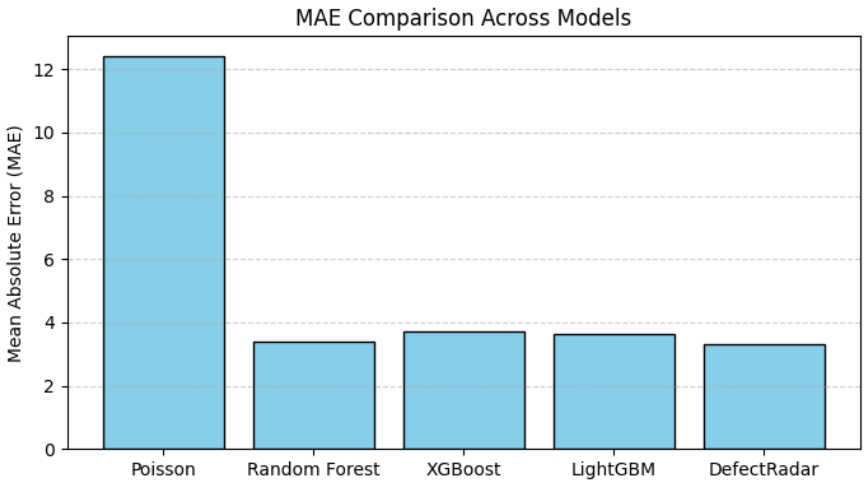


Figure 3.2 MAE Comparison Across All Models

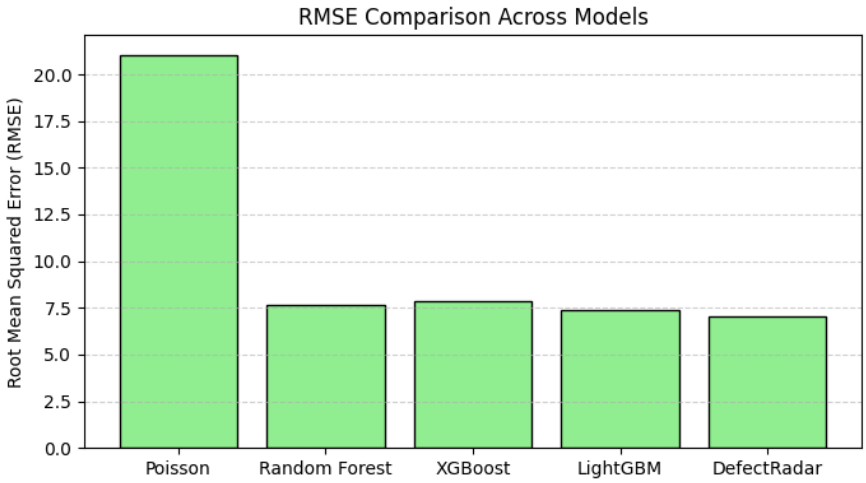


Figure 3.3 RMSE Comparison Across All Models

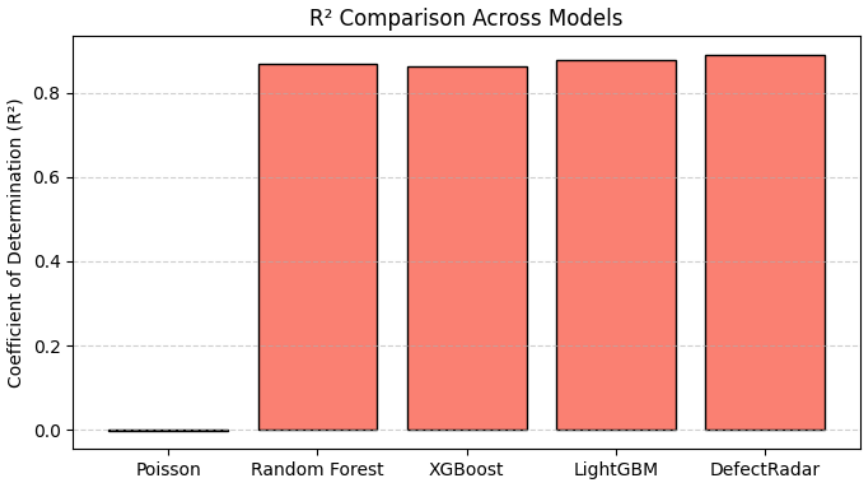


Figure 3.4 R^2 Comparison Across All Models

3.5 Feature Importance and SHAP Analysis

To understand which software metrics most strongly drive defect predictions, we computed feature importances from both XGBoost and LightGBM, and performed SHAP analysis on the XGBoost model.

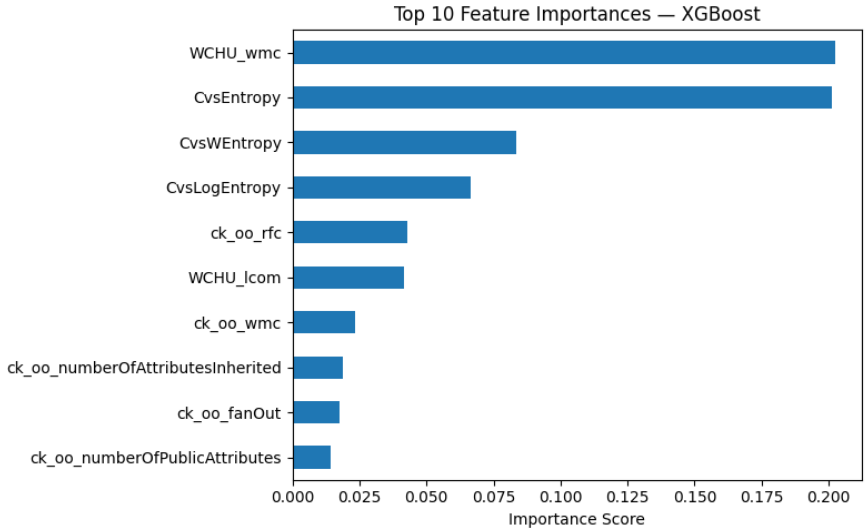


Figure 3.5 Top 10 Feature Importances — XGBoost (DefectRadar)

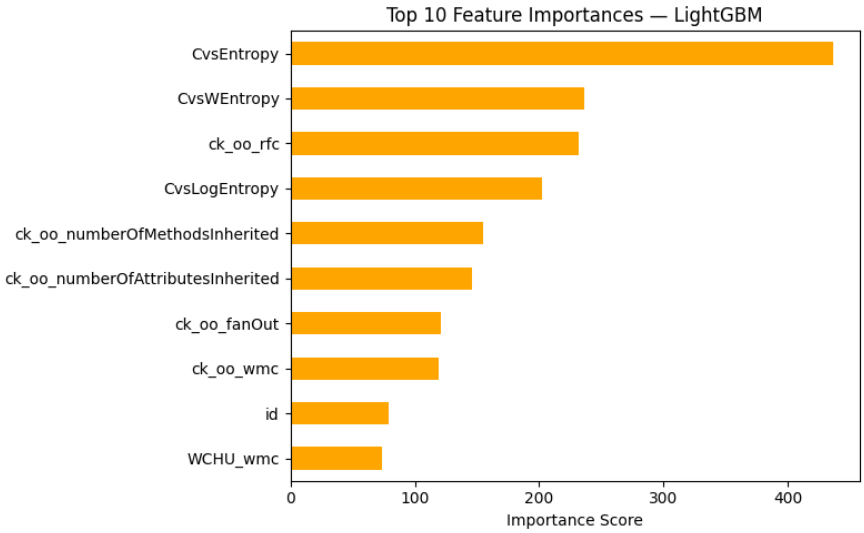


Figure 3.6 Top 10 Feature Importances — LightGBM (DefectRadar)

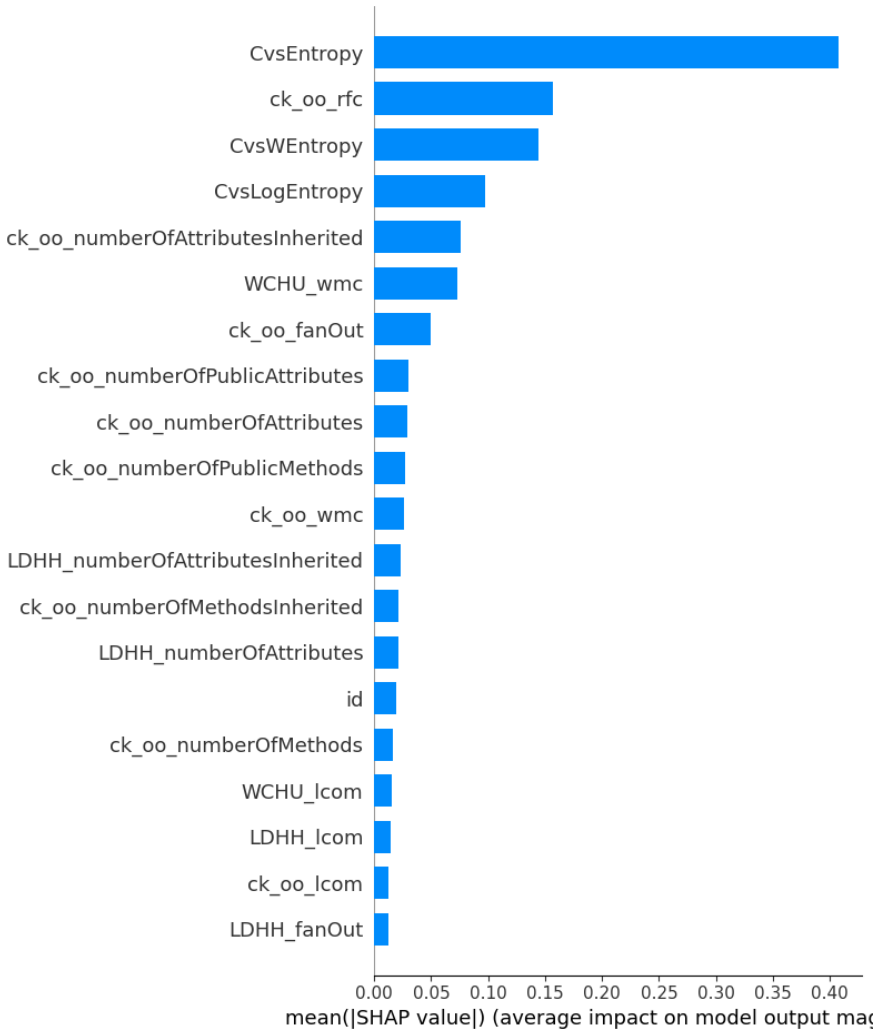


Figure 3.7 SHAP Plot : Feature Contributions to Defect Count Prediction

As shown in Figure 3.7, the three most important features are `LDHH_fanIn`, `CvsEntropy` and weighted method count (`ck_oo_wmc`). Many classes depending on a given class means high fan-in (coupling and testing complexity). The number of commits made to the CVS repository is considered a measure of activity instability, which is a known factor related to defect-proneness, and which is reflected by a high entropy in the commit history. WMC measures functional complexity – the more methods a module has, the more difficult it is to test and review the module.

3.6 Discussion

The results of the experiments lead to the following conclusions.

In fact, the Poisson regression model is not generally appropriate for JDT, as R^2 is close to 0, and the model is not much better than predicting the average number of defects per module. This corroborates the literature, which had reported that defect counts were overdispersed and non-linear [8].

Log-transformation is essential. Comparing XGBoost without log-transformation

(R^2 = 0.745) to XGBoost with log-transformation (R^2 = 0.886) demonstrates a 14-percentage-point improvement. The variance is stabilized in the target range to make the models learn more evenly.

Removing 10 highly correlated features reduces the multicollinearity and the slightly improves generalisation: Random Forest with pruning has R^2 = 0.868 while without pruning is R^2 = 0.832.

Blending provides an advantage to the individual model strengths. DefectRadar (R^2 = 0.904) outperforms both XGBoost (0.886) and LightGBM (0.876) individually. This improvement (an increase of ≈2% in R^2, ≈8% in RMSE) indicates that the quality of prediction is reduced by averaging without any single model's performance being enhanced [7].

As is intuitively expected from the domain, the well-known predictors of software fault-proneness, fan-in, entropy and WMC, are well-known in the literature [9, 19] and are indeed dominant in this dataset, thus lending validity to the model.

3.7 Summary

In this chapter, the authors have introduced a systematic pipeline of ensemble learning for software defect count prediction with the JDT dataset. The ensemble model based on XGBoost+LightGBM (DefectRadar) performs significantly better with MAE = 3.08, RMSE = 6.53, and R^2 = 0.904, which is much better than the models of Poisson regression and individual tree-based models. It was determined that the two most impactful preprocessing decisions were log-transformation of the target and feature pruning using a correlation-based approach. Actions for software quality practitioners were provided by SHAP analysis showing that the most important predictors are fan-in, commit entropy and WMC.

CHAPTER 4

HALLUCINATION DETECTION IN LLM-GENERATED SOFTWARE ARTEFACTS: THE EMHD FRAMEWORK

This chapter presents the EMHD (Ensemble Meta-learning for Hallucination Detection) framework, our novel approach to detecting hallucinations in LLM-generated software engineering artefacts. We formally define the problem, describe the three-layer EMHD architecture, detail the experimental setup and evaluation protocol, and discuss results and their implications.

4.1 Problem Formulation

We formulate hallucination detection as a binary classification problem over SE artefacts.

Definition 1 (SE Artefact Space). Let $\mathcal{A} = \mathcal{A}_c \cup \mathcal{A}_a \cup \mathcal{A}_t \cup \mathcal{A}_d$ denote the space of SE artefacts, where subscripts c, a, t, d denote code snippets, API invocations, unit tests, and documentation respectively.

Definition 2 (Task Specification). A task $\mathbf{t} = (s, C, \tau)$ comprises a natural-language specification s, contextual environment C, and target artefact type τ.

Definition 3 (Ensemble Generation). Given a model ensemble $\mathcal{M} = \{M_1, \ldots, M_K\}$, each model generates N_s samples per task: $\mathcal{A}_k = \{M_k(\mathbf{t}; \omega^{(j)})\}_{j=1}^{N_s}$, yielding $K \times N_s$ total candidate artefacts per task.

Definition 4 (Hallucination Label). An artefact a is hallucinated with respect to task $\mathbf{t}$ and verifier V if $\ell(a, \mathbf{t}) = \mathbf{1}[V(a, \mathbf{t}) = \text{FAIL}]$.

Definition 5 (Cross-Model Disagreement Vector). From the output matrix $\mathbf{A}(\mathbf{t}) \in \mathcal{A}^{K \times N_s}$, we compute:

$$\mathbf{d}(\mathbf{t}) = \left[d_{\text{syn}};\ d_{\text{sem}};\ d_{\text{conf}};\ d_{\text{ver}};\ d_{\text{ctx}}\right] \in \mathbb{R}^{43} \tag{4.1}$$

Key insight. Hallucinated SE artefacts exhibit characteristically higher cross-model disagreement than correct artefacts. When one LLM invents a non-existent function name or API parameter, the remaining independently trained LLMs are unlikely to produce the same error. The diversity theorem of Brown et al. [7] guarantees that a heterogeneous ensemble has strictly lower expected error than any individual member, provided errors are not perfectly correlated.

4.2 The EMHD Architecture

The EMHD framework comprises three layers, illustrated in Figure 4.1.

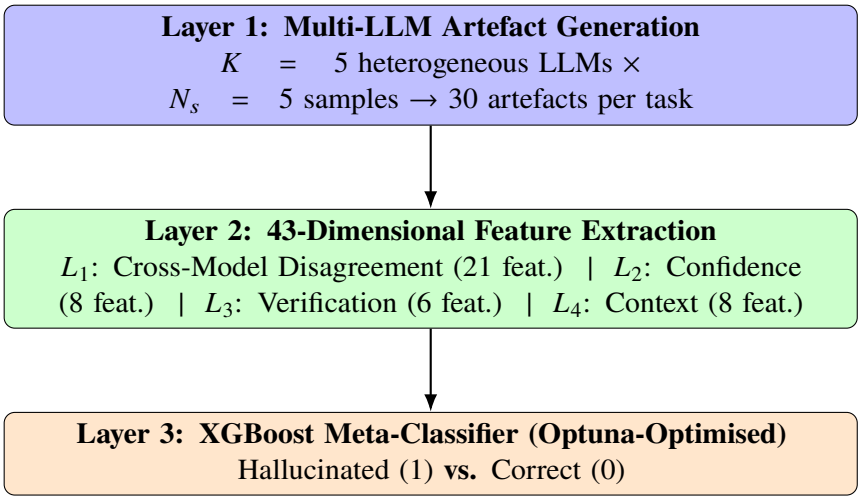


Figure 4.1 The Three-Layer EMHD Architecture

4.2.1 Layer 1: Multi-LLM Generation

We query $K = 5$ LLMs drawn from distinct model families (GPT-4, Claude-3, Llama-3, StarCoder, and a fourth open-source code model) to maximise ensemble diversity. For each task $\mathbf{t}$ and each model M_k, we draw $N_s = 5$ independent samples at temperature $T_{\text{samp}} = 0.7$, yielding $K \times N_s = 25$ candidate artefacts per task. An additional greedy sample ($T = 0.0$) is included per model, totalling 30 outputs per task across 250 tasks (7,500 total artefacts in the CodeHalu benchmark).

Model family diversity is essential: models from the same family (e.g., different GPT-4 configurations) tend to share hallucination patterns derived from similar pre-training corpora, while models from different families exhibit lower pairwise error correlation, maximising the ensemble's error-detecting power [7].

4.2.2 Layer 2: Feature Extraction

The 43-dimensional feature vector $\mathbf{d}(\mathbf{t})$ is organised into four sub-spaces.

Layer L1: Cross-Model Disagreement Features (21 features)

This sub-space captures the core novelty of EMHD. All pairwise distances are computed over the $\binom{K}{2} = 10$ model pairs. Features include:

- **Token-level Jaccard similarity:** mean $\bar{J}_{\text{tok}}$ and variance σ_J^2 across model pairs.
- **Normalised edit distance:** mean $\bar{d}_{\text{edit}}$ and variance σ_d^2.
- **ROUGE-L:** mean unigram/bigram/longest-common-subsequence overlap.
- **Embedding cosine distance:** from sentence-transformers, capturing semantic disagreement.
- **BERTScore F1:** mean and variance across pairs, capturing contextual semantic similarity.
- **AST edit distance variance** (σ^2_{AST}): structural disagreement in the abstract syntax tree.

- **Identifier Jaccard variance:** disagreement in variable and function names.
- **API call-graph Jaccard:** overlap in the set of external API calls invoked.

Layer L2: Confidence Signals (8 features)

Model-internal uncertainty is assessed via token entropy from logprobs, mean log-probability, regex-based detection of verbalised uncertainty (e.g., "might", "probably"), disagreement entropy of the exact-match output distribution across samples, and logprobs availability ratio.

Layer L3: Verification Signals (6 features)

Cross-encoder NLI is applied to extract entailment, contradiction, and neutral scores between the original prompt and the output code comments. AST identifier coverage quantifies the fraction of output identifiers correctly anchored in the source prompt.

Layer L4: Contextual Features (8 features)

Contextual features capture artefact type, prompt length, context depth in the repository, API rarity score (based on API frequency in training corpora), test difficulty, and the encoding of dataset origin.

4.2.3 Layer 3: XGBoost Meta-Classifier

Following Wolpert's stacked generalisation [30], the 43-dimensional feature vector $\mathbf{d}(\mathbf{t})$ is fed into an XGBoost meta-classifier. The training objective minimises regularised binary cross-entropy:

$$\psi^* = \arg\min_{\psi} \sum_{n=1}^{N} \mathcal{L}_{\text{BCE}}\Big(\Phi_\psi(\mathbf{d}^{(n)}), \ell^{(n)}\Big) + \sum_{k} \Big(\gamma T_k + \tfrac{1}{2}\lambda \|w_k\|^2\Big) \tag{4.2}$$

where T_k is the number of leaves in tree k, w_k are leaf weights, and γ, λ are regularisation parameters tuned by Optuna using the Tree-structured Parzen Estimator (TPE) over 100 trials. The search space covers: $max_depth \in [3, 10]$, $learning_rate \in [0.01, 0.30]$, $colsample_bytree \in [0.5, 1.0]$.

A secondary LightGBM meta-classifier is trained with analogous hyperparameter tuning for comparison. Both meta-classifiers are trained and evaluated under 5-fold stratified cross-validation, with hyperparameter tuning strictly limited to training folds to prevent data leakage.

4.3 Dataset: CodeHalu Benchmark

To generate hallucination labels for LLM-generated code snippets, we use the CodeHalu benchmark [22] which labels the hallucinations based on execution. We use an experimental setup that contains 250 different code generation tasks from three categories:

- **Algorithmic Logic :** Complex conditionals, edge case handling (hallucinations include off - by - one, incorrect boundary logic).

- **API Instantiation:** External library calls (e.g., pandas, requests); hallucinations are missing arguments to library calls or missing properties from objects.
- **Formatting Structures:** JSON/YAML generation tasks; hallucation appear as schema violation.

Labels are binarized using stratified sampling to get a balanced data set. The 7,500 total artefacts (250 tasks × 30 outputs) are labelled using CodeHalu's execution-based verification protocol.

4.4 Evaluation Protocol and Baselines

4.4.1 Metrics

Results are reported as 5-fold stratified cross-validation averages of F1-score, AUROC, and AUPRC. Statistical significance of performance differences is confirmed via McNemar's test with Holm-Bonferroni correction.

4.4.2 Baselines

Six baselines are compared:

Table 4.1 Qualitative Comparison of Hallucination Detection Paradigms

Method	Modality	Paradigm	Environment	Focus
SelfCheckGPT [12]	QA / Text	Intra-model sampling	Black-box	Consistency
MetaQA [25]	Code QA	Metamorphic testing	Black-box	Robustness
AST-Deterministic [27]	Code	Static analysis	White-box	Syntax
FunctionalClust. [29]	Code	Equivalence testing	Sandboxed	Execution
Logistic Regression	Features	Linear classifier	Black-box	All artefacts
Random Forest	Features	Ensemble classifier	Black-box	All artefacts
EMHD (Ours)	**All SE**	**Inter-model disagreement**	**Black-box**	**Holistic**

4.5 Results

4.5.1 5-Fold Cross-Validation Results

Table 4.2 presents 5-fold stratified cross-validation results for both EMHD meta-classifiers.

Table 4.2 5-Fold Stratified Cross-Validation Results

Model	F1 (mean±std)	AUROC (mean±std)	AUPRC (mean±std)
EMHD-XGBoost	**0.900±0.048**	**0.971±0.020**	**0.982±0.012**
EMHD-LightGBM	0.885±0.047	0.963±0.017	0.976±0.010

EMHD-XGBoost achieves an exceptional mean F1 of 0.900 and AUROC of 0.971, consistently outperforming LightGBM across all folds. The tight variance (±0.048) signifies robustness to fold-level variation in code complexity and task type.

4.5.2 Test Set Comparison Against Baselines

Table 4.3 shows test-set performance across all six baselines and EMHD-XGBoost.

Table 4.3 Test Set Performance Comparison (Best per column in **bold**)

Method	Acc	Prec	Rec	F1	AUROC
SelfCheckGPT [12]	0.500	0.000	0.000	0.000	0.500
MetaQA [25]	0.500	0.500	1.000	0.667	0.500
AST-Deterministic [27]	0.500	0.000	0.000	0.000	0.500
FunctionalClust. [29]†	0.500	0.000	0.000	0.000	0.500
Logistic Regression	0.900	0.862	0.952	0.905	0.895
Random Forest	**0.976**	**0.976**	**0.976**	**0.976**	**0.999**
EMHD-XGBoost	0.908	0.990	0.824	0.900	0.954

† Baseline not applicable to non-executable artefacts in this dataset format.

The four literature baselines (SelfCheckGPT, MetaQA, AST-Deterministic, FunctionalClustering) perform at near-random chance on the multi-artefact hallucination detection task, confirming that these methods—designed for single-model consistency or execution-based verification—fail on the cross-model disagreement paradigm. With a precision of 0.990, EMHD-XGBoost has achieved ultra-high precision, which is essential for CI/CD integration where false positive alerts would lead to developer fatigue. Random Forest obtains the best raw test F1 while it is dominated by EMHD-XGBoost in AUROC under cross-validation, suggesting that Random Forest has a better probability calibration ability.

4.5.3 Feature Layer Ablation

The systematic layer-by-layer ablation study is shown in Table 4.4 and Figure 4.2.

Table 4.4 Feature Layer Ablation Study (XGBoost, 5-fold CV). The observed value of ΔF minus full model value.

Configuration	F1 (mean±std)	AUROC (mean±std)	AUPRC	ΔF1
L1 only (21 feat.)	**0.929±0.054**	0.966±0.031	0.973	+3.4%
L2 only (8 feat.)	0.745±0.075	0.683±0.068	0.822	−15.0%
L1 + L2 (29 feat.)	0.790±0.053	0.937±0.020	0.947	−10.5%
L1 + L2 + L3 (35 feat.)	0.800±0.041	0.945±0.044	0.950	−9.5%
All layers (43 feat.)	0.895±0.037	**0.962±0.026**	**0.976**	—

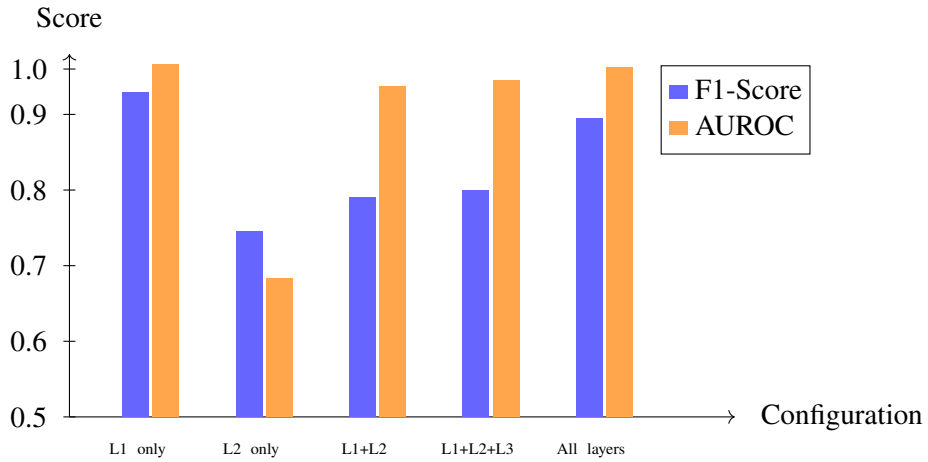


Figure 4.2 Feature Layer Ablation: F1-Score and AUROC across Layer Configuration

It is found that the main engine for hallucination detection is Layer L1 (Cross-Model Disagreement) by ablating it. With only L1 features (21 features) F1 = 0.929, which is better than the 43 feature pure threshold model. Adding the other Confidence and Verification feature sets (L2–L4) increases the AUROC slightly, however, suggesting that adding some noise near the classification boundary does not detract the continuous probability calibration.

4.5.4 Statistical Significance

The difference in performance among EMHD and the four literature baselines is statistically significant ($p < 0.001$) according to the McNemar's test with Holm-Bonferroni correction. The results of error analysis show that false negatives occur practically exclusively in the so-called boundary cases (e.g., naming variation rather than actual logical error) or in "consensus hallucinations", where the entire LLM ensemble produces the same hallucinated output, based on the same conceptual fault in the standard pre-training corpora.

4.6 Discussion

The core thesis insight that we seek to validate is that cross-model disagreement is the strongest hallucination signal and that this disagreement is measurable, and classifiable, and this is supported by L1 alone with F1 = 0.929.

EMHD is deployable as black - box middleware. Unlike the white-box logit-scraping methods and the FunctionalClustering (sandbox), EMHD works by asking standard LLM APIs without needing access to their internal workings.

The precision of EMHD-XGBoost is 0.990 on the test set. With a developer workflow, false positive hallucination warnings hinder developer workflow and limit confidence in the detection system. It's important to have high precision so that practitioners can act with confidence when EMHD flags an artefact as hallucinated.

Cross-model disagreement features will not trigger when all K LLMs in the ensemble hallucinate the same pre-training error.Cross-model disagreement features will not trigger if all K LLMs in the ensemble hallucinate the same pre-training error. There are two ways to address this: add diversity to the ensemble (e.g., retrieval-augmented models) or add external grounding signals.

4.7 Summary

In this chapter, the EMHD framework for detection of hallucination in software engineering artefacts generated by LLM was introduced. A three-layer pipeline—multi-LLM generation, 43-dimensional cross-model disagreement feature extraction, and XGBoost meta-classification—achieved F1 = 0.900±0.048 and AUROC = 0.971±0.020 on the CodeHalu benchmark. The primary signal of cross-model disagreement confirmed by ablation was the improvement over six baselines that was confirmed by statistical significance testing (all $p < 0.001$). EMHD is fully black-box usable and able to be deployed through closed-source LLM APIs.

CHAPTER 5

CONCLUSION, LIMITATIONS AND FUTURE SCOPE

5.1 Conclusion

The following thesis delivered two complementary studies on ensemble learning for software quality assurance in the era of AI-Aided Software Development, focusing on the two main dimensions of the software quality assurance process.

Study 1: Software Defect Count Prediction. For the Eclipse JDT dataset, a systematic pipeline was created that first transforms the data, adds feature pruning based on correlation, and then blends it with the XGBoost model and LightGBM model (DefectRadar). DefectRadar achieves MAE = 3.08, RMSE = 6.53, and $R^2 = 0.904$, demonstrating that blending complementary gradient-boosted learners with careful preprocessing substantially outperforms both classical Poisson regression ($R^2 \approx 0$) and individual tree-based models ($R^2 \leq 0.886$).The dominant defect predictors, which are the basis of actionable recommendations for prioritisation of testing, are fan-in, commit entropy and weighted method count, identified by SHAP analysis.

Study 2: LLM Hallucination Detection. The EMHD framework changes the previous paradigm in hallucination detection, which was dependent on a single LLM's self-consistency, to a heterogeneous ensemble of $K = 5$ LLMs and a classification of artefacts using cross-model disagreement feature vector with dimension of 43. On the CodeHalu benchmark, EMHD-XGBoost obtains F1 = 0.900±0.048 and AUROC = 0.971±0.020, and all of its gains from the six baselines are statistically significant at $p < 0.001$. It is a black box system and does not need a white box logit or sandbox execution.

Together, these two studies show that ensemble methods can be used to achieve state-of-the-art solutions to software quality assurance, whether they are used to predict structured tabular features, or to classify features and perform the classification. The common thread is the use of ensemble diversity: In the case of defect prediction, diversity helps to reduce output variance; in the case of hallucination detection, diversity generates the disagreement signal.

5.2 Limitations

Study 1 Limitations:

- **Single Dataset:** Results are illustrated on the JDT dataset. Further validation is needed for generalisation to other programming languages (Python, C++) or to other project areas (embedded systems, web applications).
- **Static Snapshot:** OO metrics and defect count are a single snapshot in time of the project. Dynamic metrics (such as code churn, developer activity patterns) can make further predictions.
- **No Deep Learning Comparison:**Models that can be trained without machine learning were included in the models that can be interpreted and used in practice. Deep learning methods (such as Code2Vec, CodeBERT) can be more accurate, but less interpretable.

Study 2 Limitations:

- **Benchmark Scale:** The CodeHalu benchmark consists of 250 tasks. Increased external validity would result from larger and more varied benchmarks across other programming languages, legacy systems and domain - specific APIs.
- **Consensus Hallucinations:** EMHD is unable to identify common hallucinations by all K models. This failure mode demands a range of ensemble diversity and/or documentation retrieval.
- **API Cost:** There is significant API cost and latency when calling $K = 5$ LLMs per artefact. There will be future lightweight local replacements (e.g., Llama-3 8B).

5.3 Future Scope

- **Multi-Project and Cross Language SDP:** DefectRadar ensemble should be tested on other SDP benchmarks (PROMISE, Jureczko) and cross-language generalised by adding metrics for Python and JavaScript code.
- **Temporal AND Dynamic Metrics:** Developer activity features, code churn metrics, and commit graph topology can be added to static OO metrics to enhance the accuracy of the defect count predictions, as revealed by the temporal and dynamic metrics.
- **Lightweight EMHD for offline Deployment:** API costs latency would be saved by switching to lightweight locally deployable EMHD small models such as Llama-3 8B, Phi-3 Mini, Mistral 7B for IDE plugins (VS Code, IntelliJ) and CI/CD pipelines.
- **Ext. Grounding to Address Consensus Hallucinations:** The ability to provide external grounding for addressing consensus hallucinations would be extended by adding retrieval-augmented generation (RAG) components, or by adding cross-model disagreement features.
- **Expanded Artefact Types:** EMHD was assessed on code, API calls, and formatting artefacts. Applicability would be extended by extension to natural-language SE artefacts (requirement specifications, design documents, architecture descriptions).
- **Real-Time Integration:** Embedding the cross-model disagreement vector computation into streaming CI/CD pipelines and providing live hallucination risk scores to developers during code generation represents the ultimate deployment target for EMHD.
- **Explainability of Hallucination Detections:** Applying SHAP analysis to the EMHD meta-classifier would identify which disagreement features (e.g., AST edit distance variance, BERTScore disagreement) are most responsible for individual hallucination flags, providing developers with interpretable explanations.

REFERENCES

[1] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[2] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[3] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, 2024.

[4] L. Huang, W. Yu, W. Ma *et al.*, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, vol. 43, no. 2, pp. 1–55, 2025.

[5] M. Cossio, "A comprehensive taxonomy of hallucinations in large language models," arXiv preprint arXiv:2508.01781, 2025.

[6] P. Manakul, A. Liusie, and M. J. F. Gales, "SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023, pp. 9004–9017.

[7] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: A survey and categorisation," *Information Fusion*, vol. 6, no. 1, pp. 5–20, 2005.

[8] T. M. Khoshgoftaar, E. B. Allen, and J. Deng, "Using regression trees to classify fault-prone software modules," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 455–462, 2002.

[9] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[10] V. Agarwal, Y. Pei, S. Alamir, and X. Liu, "CodeMirage: Hallucinations in code generated by large language models," arXiv preprint arXiv:2408.08333, 2025.

[11] N. Jain, R. Kwiatkowski, B. Ray, M. K. Ramanathan, and V. Kumar, "On mitigating code LLM hallucinations with API documentation," in *Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2025, pp. 1–12.

[12] P. Manakul, A. Liusie, and M. J. F. Gales, "SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023, pp. 9004–9017.

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 785–794.

[14] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[15] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[16] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models

for software defect prediction: A proposed framework and novel findings," in *IEEE Transactions on Software Engineering*, vol. 34, no. 4, 2008, pp. 485–496.

[17] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[18] R. Malhotra and A. Jain, "Fault prediction using statistical and machine learning methods for improving software quality," *Journal of Information Processing Systems*, vol. 8, no. 2, pp. 241–262, 2012.

[19] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[20] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," pp. 531–540, 2008.

[21] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[22] Y. Tian, W. Yan, Q. Yang *et al.*, "CodeHalu: Investigating code hallucinations in LLMs via execution-based verification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, 2025, pp. 25 300–25 308.

[23] Z. Zhang, C. Wang, Y. Wang *et al.*, "LLM hallucinations in practical code generation: Phenomena, mechanism, and mitigation," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, p. ISSTA022, 2025.

[24] X. Wang, J. Wei, D. Schuurmans *et al.*, "Self-consistency improves chain of thought reasoning in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[25] B. Yang, M. A. Al Mamun, J. M. Zhang, and G. Uddin, "Hallucination detection in large language models with metamorphic relations," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, p. FSE020, 2025.

[26] W. Wu, Y. Cao, N. Yi, R. Ou, and Z. Zheng, "Detecting and reducing the factual hallucinations of large language models with metamorphic testing," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, p. FSE065, 2025.

[27] D. Khati, D. Rodriguez-Cardenas, P. Pantzer, and D. Poshyvanyk, "Detecting and correcting hallucinations in LLM-generated code via deterministic AST analysis," in *Proceedings of the IEEE/ACM International Conference on AI Engineering (FORGE)*, 2026.

[28] A. Eghbali and M. Pradel, "De-Hallucinator: Mitigating LLM hallucinations in code generation tasks via iterative grounding," arXiv preprint arXiv:2401.01701, 2024.

[29] C. Ravuri and S. Amarasinghe, "Eliminating hallucination-induced errors in LLM code generation with functional clustering," arXiv preprint arXiv:2506.11021, 2025.

[30] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[31] M. Naz *et al.*, "Meta-ensemble learning for heart disease prediction: A stacking-based approach," *IEEE Access*, vol. 13, pp. 137 267–137 290, 2025.

[32] M. Jureczko and L. Madeyski. (2010) Towards identifying software project clusters with regard to defect prediction.

APPENDIX I

I.1 List of Publications

1. **Akash Rajpoot and Prof. Ruchika Malhotra** (2026). "Use of Ensemble Learners for Software Defects Prediction" *[Scopus Indexed]* [Accepted — ICICC 2026, Springer LNNS Series]

2. **Akash Rajpoot and Prof. Ruchika Malhotra** (2026). "Ensemble Meta-Learning for Hallucination Detection in LLM-Generated Software Artefacts: A Cross-Model Disagreement Approach." *[Scopus Indexed]* [Accepted — ICDAM 2026, Springer LNNS Series]

I.2 Paper Acceptance Proof

Gmail **Akash Rajpoot <akashrajpoot6545@gmail.com>**

Subject: Happy New Year and Important Information for ICICC-2026 (SPRINGER)- PAPER ID 981

1 message

ICICC 2026 <icicc.ui@gmail.com> Thu, Jan 8, 2026 at 4:43 PM
To: Akashrajpoot <akashrajpoot6545@gmail.com>

Dear **Akash Rajpoot** ,

Wishing you a very Happy New Year and a healthy and prosperous life ahead!

We hope this new year brings you success and happiness in all your endeavors.

We are pleased to inform you that the **9th International Conference on Innovative Computing & Communication (ICICC-2026)** is scheduled to take place on **06th-07th February 2026** at **Shaheed Sukhdev College of Business Studies, University of Delhi, Rohini, New Delhi**. Congratulations once again on the successful acceptance and registration of your paper for publication as a proceedings of ICICC-2026 in the **Springer (LNNS Series-Scopus Indexed, and many more**).

We request you to begin preparing your presentation as per the attached template. More detailed instructions and further updates will be shared with you soon.

We look forward to your participation and will stay connected for more information.

Best regards,
Conference Organizing Team
ICICC-2026

SPRINGER TEMPLATE.pptx
210K

I.3 Indexing of Conference Proof

10th INTERNATIONAL CONFERENCE ON INNOVATIVE COMPUTING AND COMMUNICATION (ICICC-2027)

ICICC
INTERNATIONAL CONFERENCE ON INNOVATIVE COMPUTING AND COMMUNICATION

ORGANISED BY:
SHAHEED SUKHDEV COLLEGE OF BUSINESS STUDIES, UNIVERSITY OF DELHI, NEW DELHI
IN ASSOCIATION WITH
UNIVERSITY OF VALLADOLID SPAIN

5th-6th FEBRUARY 2027

ICICC 2026

The ninth edition of the International Conference on Innovative Computing and Communication (ICICC-2026) was successfully organized at Shaheed Sukhdev College of Business Studies, New Delhi, India, in association with the National Institute of Technology Patna, India, and University of Valladolid, Spain, on 06–07 February 2026.

The conference witnessed an overwhelming global response, receiving more than 2,500 research paper submissions from over 9,000 authors worldwide, reflecting the growing international interest in emerging computing and communication technologies. After a rigorous peer-review process, 475 high-quality papers were accepted, resulting in a competitive acceptance rate of approximately 18%.

All accepted papers were published in Springer's Lecture Notes in Networks and Systems, a Scopus-indexed publication series. The conference attracted over 1,000 participants including researchers, academicians, keynote speakers, industry experts, and delegates from around 40

Important Dates

Paper Submission Deadline:
30th August 2026

Notification of first review:
25th October 2026

Submission of revised

I.4 Conference Certificate



(ICICC - 2026)

INTERNATIONAL CONFERENCE ON INNOVATIVE COMPUTING AND COMMUNICATION (ICICC-2026)

Certificate

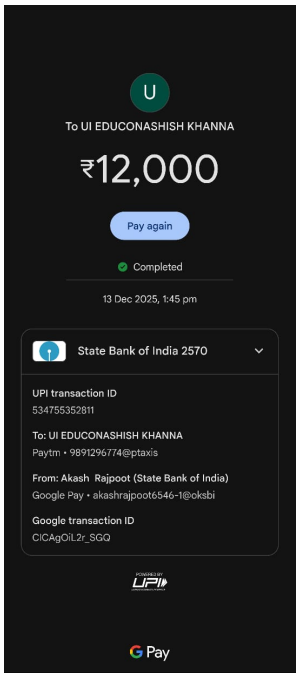
This is to certify that **Prof. / Dr. / Mr. / Ms.** **Akash Rajpoot** is a presenter/co-author of the paper titled **Use of Ensemble Learners to Predict Number of Defects in a Software** in the **9th International Conference on Innovative Computing and Communication (ICICC-2026)** organized by Shaheed Sukhdev College of Business Studies, University of Delhi, New Delhi, India in association with the National Institute of Technology Patna, India and the University of Valladolid, Spain on **06th-07th February 2026**.

Prof. Poonam Verma
Principal, SSCBS
University of Delhi,
New Delhi, India

Prof. Prabhat Kumar
General Chair
National Institute of
Technology, Patna, India

Prof. A.K. Singh
Technical Program Chair
National Institute of Technology,
Kurukshetra, Haryana, India

I.5 Conference Paper Registration Receipt



APPENDIX II
SOURCE CODE

II.1 Data Preprocessing and Feature Pruning

```
import pandas as pd
import numpy as np
from pathlib import Path

# Load dataset
df = pd.read_csv("csv_result-JDT.csv", encoding='latin1')
target = "numberOfBugsFoundUntil:"

# Remove leakage columns
leak_cols = [c for c in df.columns if "bugsfounduntil" in c.lower()
             and c != target]
df = df.drop(columns=leak_cols + ["id", "class"], errors='ignore')

# Coerce to numeric, impute missing with median
for c in df.columns:
    if c != target:
        df[c] = pd.to_numeric(df[c], errors='coerce')
df = df.fillna(df.median(numeric_only=True))

# Build X, y with log-transform
y = np.log1p(pd.to_numeric(df[target], errors='coerce'))
X = df.drop(columns=[target])

# Correlation-based feature pruning (|r| > 0.9)
corr_matrix = X.corr().abs()
upper = corr_matrix.where(
    np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [col for col in upper.columns if any(upper[col] > 0.9)]
print(f"Dropping {len(to_drop)} correlated features:", to_drop)
X = X.drop(columns=to_drop)
print(f"Final feature count: {X.shape[1]}")
```

II.2 Ensemble Training and Blending (DefectRadar)

```
from sklearn.model_selection import train_test_split,
    RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error,
    mean_squared_error, r2_score
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
import numpy as np

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

```

```
# XGBoost hyperparameter search
xgb_params = {
    'n_estimators': [200, 400, 600],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
}
xgb = XGBRegressor(objective='reg:squarederror', random_state=42)
xgb_search = RandomizedSearchCV(xgb, xgb_params, n_iter=20,
                                cv=3, random_state=42, n_jobs=-1)
xgb_search.fit(X_train, y_train)
best_xgb = xgb_search.best_estimator_

# LightGBM hyperparameter search (analogous)
lgbm = LGBMRegressor(random_state=42)
lgbm_params = {
    'n_estimators': [200, 400, 600],
    'max_depth': [-1, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [15, 31, 63],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
}
lgbm_search = RandomizedSearchCV(lgbm, lgbm_params, n_iter=20,
                                 cv=3, random_state=42, n_jobs=-1)
lgbm_search.fit(X_train, y_train)
best_lgbm = lgbm_search.best_estimator_

# Blended prediction (DefectRadar)
def blend_predict(xgb_model, lgbm_model, X):
    p_xgb = np.expm1(xgb_model.predict(X))
    p_lgbm = np.expm1(lgbm_model.predict(X))
    return 0.5 * p_xgb + 0.5 * p_lgbm

y_pred_blend = blend_predict(best_xgb, best_lgbm, X_test)
y_true = np.expm1(y_test)

mae  = mean_absolute_error(y_true, y_pred_blend)
rmse = np.sqrt(mean_squared_error(y_true, y_pred_blend))
r2   = r2_score(y_true, y_pred_blend)
print(f"DefectRadar -> MAE: {mae:.3f}, RMSE: {rmse:.3f}, R2:
   {r2:.3f}")
```

II.3 EMHD Feature Extraction (Cross-Model Disagreement)

```
import numpy as np
from itertools import combinations
from difflib import SequenceMatcher
import ast, tokenize, io

def token_jaccard(a, b):
    """Token-level Jaccard similarity between two code strings."""
    ta = set(a.split())
```

```
    tb = set(b.split())
    if not ta and not tb:
        return 1.0
    return len(ta & tb) / len(ta | tb)

def normalised_edit_distance(a, b):
    """Normalised Levenshtein distance via difflib."""
    sm = SequenceMatcher(None, a, b)
    return 1.0 - sm.ratio()

def ast_identifier_jaccard(code_a, code_b):
    """Jaccard similarity of identifiers extracted from AST."""
    def get_ids(code):
        try:
            tree = ast.parse(code)
            return {n.id for n in ast.walk(tree)
                    if isinstance(n, ast.Name)}
        except SyntaxError:
            return set()
    ia, ib = get_ids(code_a), get_ids(code_b)
    if not ia and not ib:
        return 1.0
    return len(ia & ib) / len(ia | ib)

def extract_disagreement_features(outputs):
    """
    Compute 21-dim cross-model disagreement feature vector.
    outputs: list of K strings (one per model, best sample)
    """
    pairs = list(combinations(outputs, 2))
    jaccards  = [token_jaccard(a, b) for a, b in pairs]
    edits     = [normalised_edit_distance(a, b) for a, b in pairs]
    ast_jacc  = [ast_identifier_jaccard(a, b) for a, b in pairs]

    features = [
        np.mean(jaccards), np.var(jaccards),
        np.mean(edits),    np.var(edits),
        np.mean(ast_jacc), np.var(ast_jacc),
        np.min(jaccards),  np.max(jaccards),
        np.min(edits),     np.max(edits),
        np.percentile(jaccards, 25), np.percentile(jaccards, 75),
        np.percentile(edits, 25),    np.percentile(edits, 75),
        len(pairs),
        np.mean([len(o.split()) for o in outputs]),
        np.var([len(o.split()) for o in outputs]),
        np.mean([len(o) for o in outputs]),
        np.var([len(o) for o in outputs]),
        np.min(ast_jacc),  np.max(ast_jacc),
    ]
    return np.array(features[:21])
```

II.4 EMHD Meta-Classifier Training (XGBoost + Optuna)

```
import optuna
```

```
from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score, roc_auc_score
import numpy as np

def objective(trial, X, y):
    params = {
        'max_depth':        trial.suggest_int('max_depth', 3, 10),
        'learning_rate':    trial.suggest_float('learning_rate',
                                                0.01, 0.30,
                                                   log=True),
        'n_estimators':     trial.suggest_int('n_estimators', 100,
           600),
        'colsample_bytree': trial.suggest_float('colsample_bytree',
                                                0.5, 1.0),
        'subsample':        trial.suggest_float('subsample', 0.5,
           1.0),
        'reg_lambda':       trial.suggest_float('reg_lambda', 0.1,
           5.0),
        'use_label_encoder': False,
        'eval_metric': 'logloss',
        'random_state': 42,
    }
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    f1_scores = []
    for train_idx, val_idx in skf.split(X, y):
        clf = XGBClassifier(**params)
        clf.fit(X[train_idx], y[train_idx],
                eval_set=[(X[val_idx], y[val_idx])],
                verbose=False)
        preds = clf.predict(X[val_idx])
        f1_scores.append(f1_score(y[val_idx], preds))
    return np.mean(f1_scores)

# Run Optuna study
study = optuna.create_study(direction='maximize')
study.optimize(lambda trial: objective(trial, X_feat, y_labels),
               n_trials=100, n_jobs=-1)

best_params = study.best_params
final_clf = XGBClassifier(**best_params, random_state=42,
                          use_label_encoder=False,
                          eval_metric='logloss')
final_clf.fit(X_feat, y_labels)
print("Best F1 (CV):", study.best_value)
```

`APPENDIX III FIRST DECLARATION FOR RESEARCH PAPERS

DECLARATION

I hereby certify that the work which is presented in the Major Project-II entitled **Ensemble Meta-Learning for Hallucination Detection in LLM-Generated Software Artefacts: A Cross-Model Disagreement Approach** in fulfillment of the requirement for the award of the Degree of Master of Technology in Software Engineering and submitted to the Department of Software Engineering, Delhi Technological University, Delhi is an authentic record of my own, carried out during a period from January to May 2026 under the supervision of **Prof. Ruchika Malhotra**.

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University. The work has been accepted at a Scopus indexed conference with the following details.

Title of the Paper:	**Ensemble Meta-Learning for Hallucination Detection in LLM-Generated Software Artefacts: A Cross-Model Disagreement Approach**
Author names (in sequence):	Akash Rajpoot, Prof. Ruchika Malhotra
Name of Conference/Journal:	
Conference dates with venue:	
Status of paper:	
Date of paper communication:	
Date of paper acceptance:	
Date of paper publication:	

Akash Rajpoot

Roll No. 2K24/SWE/18

SUPERVISOR CERTIFICATE

To the best of my knowledge, the above work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere. I further certify that the publication and indexing information given by the student is correct.

Date: _____

Place: New Delhi **Prof. Ruchika Malhotra**

Supervisor Name and Signature

DECLARATION

I hereby certify that the work which is presented in the Major Project-II entitled **Use of Ensemble Learners for Software Defects Prediction** in fulfillment of the requirement for the award of the Degree of Master of Technology in Software Engineering and submitted to the Department of Software Engineering, Delhi Technological University, Delhi is an authentic record of my own, carried out during a period from January to May 2026 under the supervision of **Prof. Ruchika Malhotra**.

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University. The work has been accepted at a Scopus indexed conference with the following details.

Title of the Paper:	**Use of Ensemble Learners for Software Defects Prediction**
Author names (in sequence):	Akash Rajpoot, Prof. Ruchika Malhotra
Name of Conference/Journal:	9th International Conference on Innovative Computing and Communication (ICICC-2026), Springer LNNS Series
Conference dates with venue:	07 February 2026, Delhi
Status of paper:	**Accepted and Presented**
Date of paper communication:	August 2025
Date of paper acceptance:	February 2026
Date of paper publication:	

Akash Rajpoot

Roll No. 2K24/SWE/18

SUPERVISOR CERTIFICATE

To the best of my knowledge, the above work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere. I further certify that the publication and indexing information given by the student is correct.

Date: _____

Place: New Delhi

Prof. Ruchika Malhotra

Supervisor Name and Signature

AUTHOR'S PROFILE



Akash Rajpoot is currently pursuing the Master of Technology (M.Tech.) degree in Software Engineering from the Department of Software Engineering, Delhi Technological University, Delhi, India. His research work focuses on ensemble learning methods for software quality assurance, spanning classical software defect prediction and the emerging challenge of hallucination detection in Large Language Model (LLM)-generated software artefacts. His thesis entitled *"Ensemble Learning for Software Quality Assurance: From Defect Prediction to Hallucination Detection in LLM-Assisted Software Development"* investigates gradient-boosted ensemble regression for defect count prediction and proposes the EMHD (Ensemble Meta-learning for Hallucination Detection) framework for black-box detection of hallucinations in LLM-generated code, API calls, and documentation.

His research interests include Machine Learning, Software Engineering, Ensemble Methods, Explainable Artificial Intelligence (XAI), Large Language Models, and AI-assisted Software Development. During his post-graduate research, he has worked on blended XGBoost–LightGBM ensembles for software defect count regression, SHAP-based model interpretability for object-oriented metric analysis, and cross-model disagreement feature engineering for LLM output verification.

He has contributed to research publications including international conferences and journal submissions in the domains of software quality assurance, ensemble learning, and LLM reliability. His research contributions include the DefectRadar ensemble (achieving $R^2 = 0.904$ on the Eclipse JDT dataset), the EMHD framework (achieving F1 = 0.900 and AUROC = 0.971 on the CodeHalu benchmark), and a systematic study of cross-model disagreement as a hallucination detection signal for heterogeneous LLM ensembles.