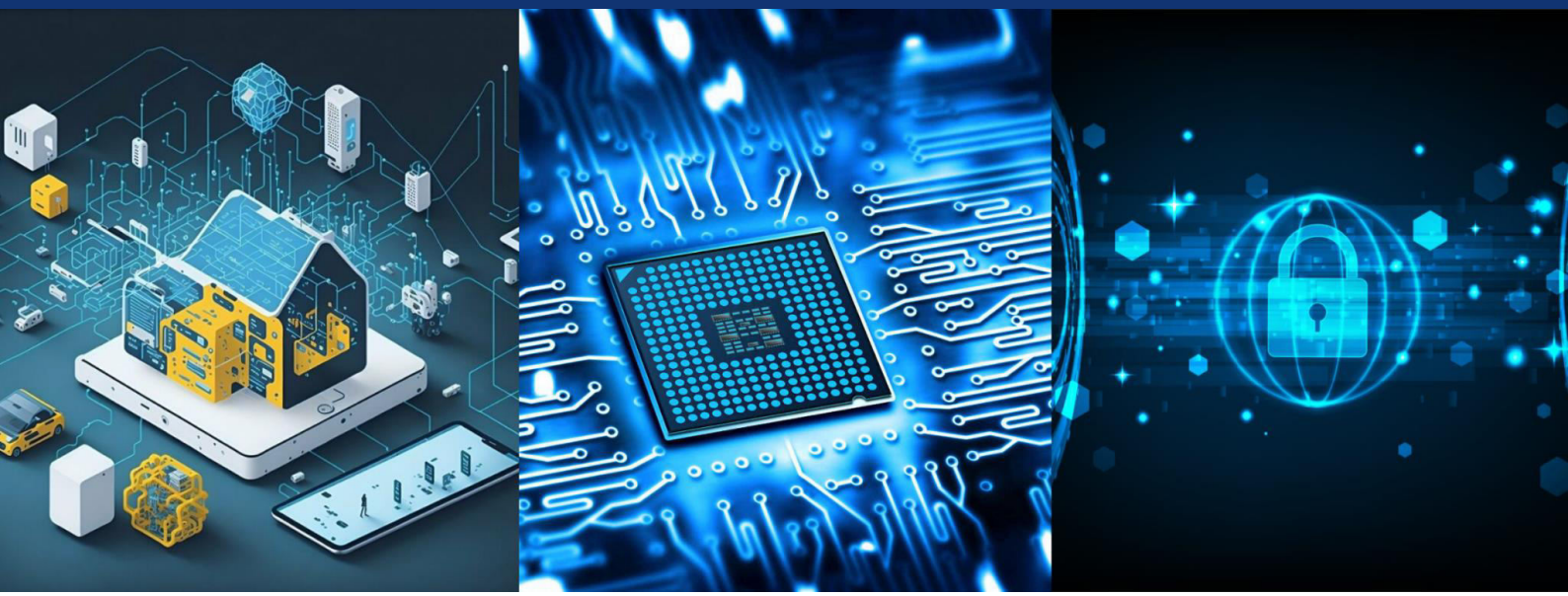
IJIRCCE

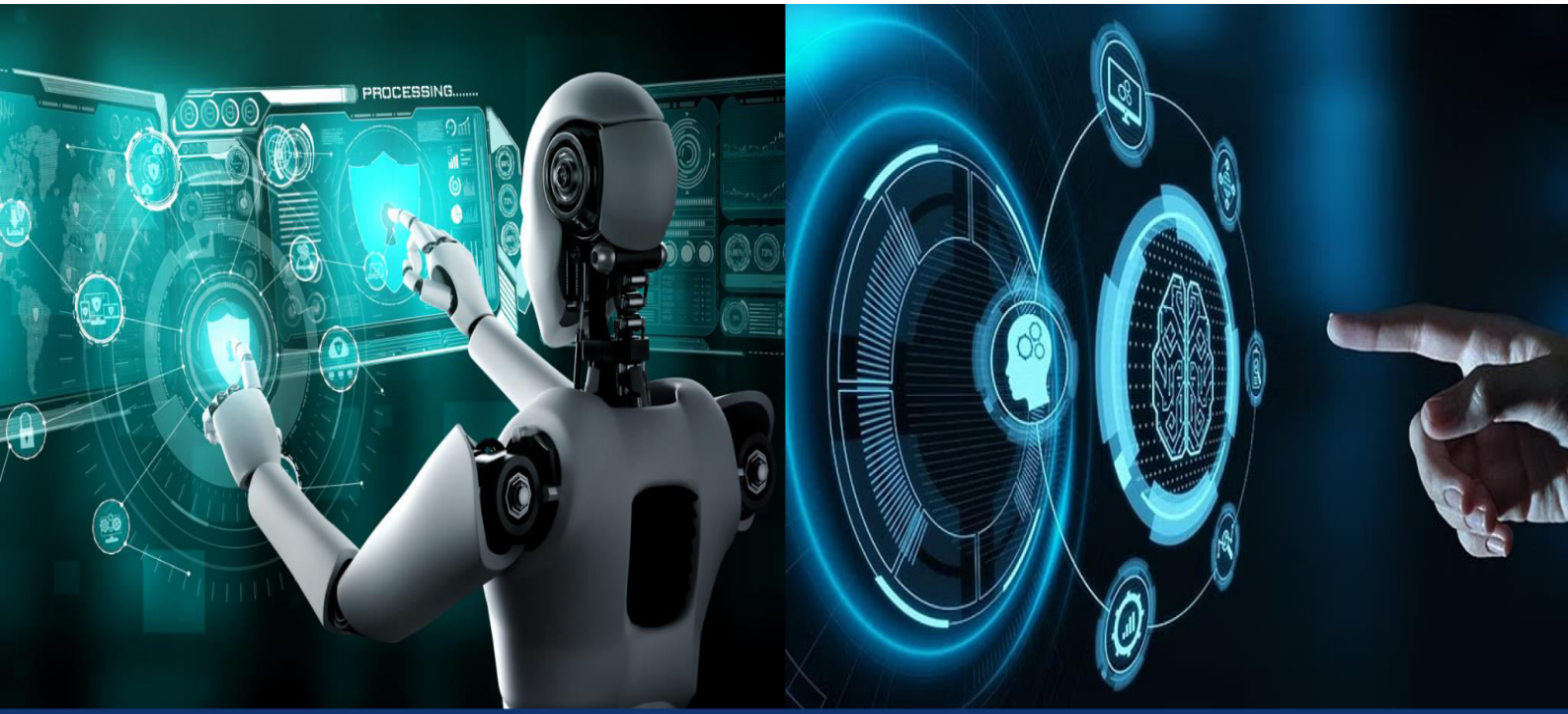
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 6, June 2026

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406090

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Software Defect Prediction using Lightweight Convolutional Neural Networks (LCNN) and Explainable AI (XAI)

Dr. T.V.S Sriram[1], P. Prasad[2], B.Chandini[3]

Sr. Asst. Professor, Dept. of MCA, NSRIT, Visakhapatnam, AP, India[1]

Asst. Professor, Dept. Of MCA, NSRIT, Visakhapatnam, AP, India[2]

Dept. of MCA, NSRIT, Visakhapatnam, AP, India[3]

ABSTRACT: Software defects continue to pose a critical challenge in mission-critical systems, with the global cost of software bugs exceeding USD 1.1 trillion annually. Traditional machine learning approaches for Software Defect Prediction (SDP) treat code metrics as flat feature vectors, failing to capture the spatial correlations between different metric groups. This paper presents LCNN, a Lightweight Convolutional Neural Network system that introduces 2D Spatial Feature Extraction — transforming 20 NASA CM1 software metrics into a 4×5 spatial grid ("Metric Image") — enabling a 2D-Residual CNN to learn hidden cross-metric dependencies invisible to tabular models. The NASA CM1 dataset (498 original samples) is expanded to 7,000 balanced samples using combined SMOTE and bootstrapped oversampling. The proposed LCNN 2D-CNN achieves a Test Accuracy of 94.50%, ROC-AUC of 0.976, and Defect Recall of 99.1% — outperforming all classical ML baselines. Explainable AI is integrated via LIME for per-prediction explanations and SHAP for global feature importance validation. The complete system is deployed as a Flask web dashboard with real-time inference, dynamic confidence bars, and embedded LIME visualizations.

KEYWORDS: Software Defect Prediction, Lightweight CNN, 2D Spatial Feature Extraction, Explainable AI, LIME, SHAP, NASA CM1, SMOTE, Flask Dashboard, Deep Learning.

I. INTRODUCTION

Software defects have become an essential challenge in modern software engineering and quality assurance systems as software complexity continues to increase with technological advancement. However, most existing systems still rely on manual code inspection or simple rule-based static analysis tools, which provide limited insights and lack predictive capabilities. This often leads to delayed defect detection, inefficient resource allocation, and significantly increased costs in the software development lifecycle.

The economic impact of software defects is staggering — NASA lost a USD 327.6 million Mars Climate Orbiter spacecraft in 1999 due to a simple unit conversion error, illustrating that in mission-critical software systems, defect prediction is a safety-critical imperative, not merely a quality concern.

To address these challenges, this project develops an LCNN-based Software Defect Prediction System that analyzes NASA spacecraft software modules using 20 Halstead complexity and McCabe cyclomatic complexity metrics, transformed into a 2D spatial feature grid, to predict whether a software module is defective or safe. The system integrates Explainable AI (LIME and SHAP) to help developers understand exactly which code properties drove each prediction, and is deployed as a complete Flask web dashboard for real-time use.

II. LITERATURE SURVEY

Software Defect Prediction has gained significant attention in recent years due to increasing software complexity and the need for efficient quality assurance systems. Many researchers have developed different machine learning and deep learning techniques to predict defect occurrences.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

A study on machine learning approaches for software defect prediction by Menzies et al. (2007) demonstrated that even simple learners such as Naive Bayes could achieve competitive SDP performance on NASA MDP datasets — the same data repository used in this project — while highlighting that class imbalance handling is the most critical preprocessing decision.

Lessmann et al. (2008) conducted comprehensive benchmarking of 22 classification algorithms for SDP, finding that ensemble methods such as Random Forest and Gradient Boosting consistently achieved strong performance but no single algorithm dominated across all benchmarks. Their study established the baseline performance levels against which the proposed LCNN system is evaluated.

Wang et al. (2016) published a seminal study demonstrating the application of Deep Belief Networks (DBNs) to SDP using deep representation learning, achieving significant accuracy improvements over handcrafted feature-based methods on NASA MDP and PROMISE datasets, establishing the potential of deep learning for SDP.

Li et al. (2017) extended deep learning to CNN-based defect detection, representing code modules as feature matrices and applying convolutional filters to capture local metric interaction patterns. Their CNN model achieved state-of-the-art performance but employed large, computationally expensive architectures without interpretability mechanisms.

Ribeiro et al. (2016) introduced LIME (Local Interpretable Model-agnostic Explanations), a widely adopted XAI technique that explains individual classifier predictions by constructing locally faithful linear approximations — integrated as a core feature in the proposed dashboard.

Lundberg and Lee (2017) introduced SHAP (SHapley Additive exPlanations), a unified framework grounded in cooperative game theory for theoretically consistent feature attribution, which is used in this project for global feature importance validation.

Overall, the literature survey shows that machine learning and deep learning techniques play a crucial role in software defect prediction systems. Most existing studies focus on sequential feature vectors, standard CNN architectures, or classical ML algorithms. However, there is still a need for systems that capture 2D spatial correlations between software metric groups, combined with integrated explainability and user-friendly dashboards. This project aims to develop a practical defect prediction system that addresses all these gaps.

Author & Year	Method	Dataset	Accuracy / AUC	Limitation
Menzies et al. (2007)	Naive Bayes, ML comparison	NASA MDP	~70–75%	No deep learning; no XAI
Lessmann et al. (2008)	22 ML classifiers	NASA MDP, AEEEM	~78%	No CNN; no explainability
Wang et al. (2016)	Deep Belief Network	NASA MDP, PROMISE	~85%	Heavyweight; no XAI
Li et al. (2017)	Standard CNN	PROMISE	~87%	No 2D spatial; no XAI
Ribeiro et al. (2016)	LIME (XAI technique)	Multiple domains	XAI only	Base model accuracy not addressed
Lundberg & Lee (2017)	SHAP (XAI technique)	Multiple domains	XAI only	Needs deep net adaptation
This Project (2025)	LCNN 2D-Residual + LIME + SHAP	NASA CM1 (7,000)	94.50% / 0.976	C-language metrics only

Table I: Comparative Summary of Related Works

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406090

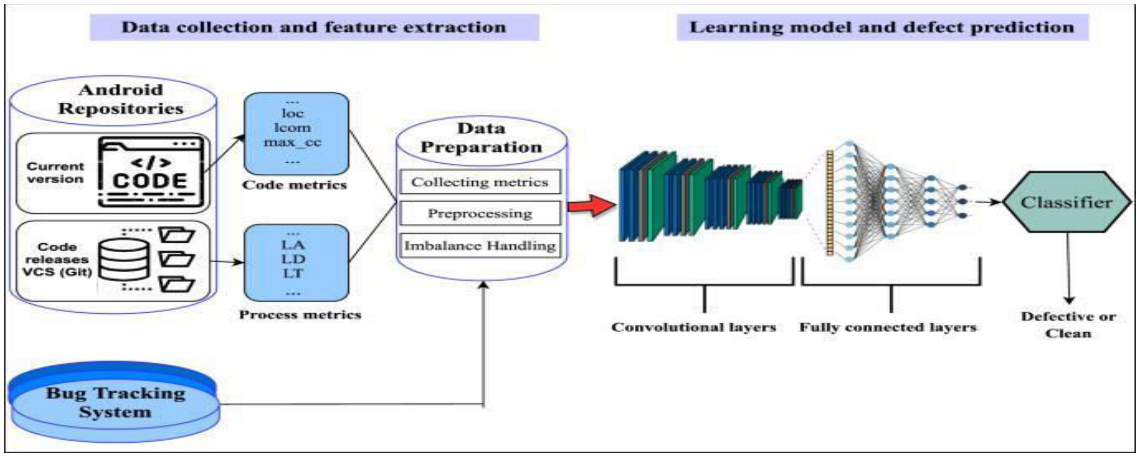
www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

III. PROPOSED SYSTEM

The proposed system consists of five integrated modules:



A. Data Collection and Dataset

The primary dataset is NASA CM1 — a real NASA Spacecraft Instrument Science System written in C, available from the PROMISE Software Engineering Repository. The original dataset contains 498 function-level modules with 20 Halstead complexity and McCabe cyclomatic complexity metrics, labeled as defective or safe. The severe class imbalance (90% safe, 10% defective) is addressed by expanding to 7,000 balanced samples using SMOTE + bootstrapped oversampling, achieving a 50/50 class distribution. Supplementary benchmarks include JM1 (9,593 samples), KC1 (2,109 samples), and PC1 (1,109 samples).

B. Data Preprocessing

Loads NASA CM1 ARFF dataset and converts to DataFrame
Removes duplicate records and imputes missing values with column medians
Applies SMOTE (k_neighbors=5) to generate synthetic defective module samples
Applies bootstrapped oversampling to expand to 7,000 balanced samples
Normalizes all features using StandardScaler (fitted on training set only)
Saves StandardScaler as models/scaler.pkl for Flask real-time inference
Stratified 70/15/15 train/validation/test split

C. Core Innovation — 2D Spatial Feature Extraction

The central innovation of this project is the transformation of the conventional 1D software metric vector (20×1) into a 4×5 two-dimensional spatial grid — treating the feature set as a "Metric Image." This is implemented using numpy.reshape(-1, 4, 5, 1). The CNN's 3×3 convolutional filters slide across both rows and columns simultaneously, learning spatial dependencies between related metric groups — for example, the interaction between Halstead Effort, Cyclomatic Complexity, and Lines of Code — that are invisible to tabular models and 1D sequential approaches.

D. LCNN Model Architectures

Two complementary CNN architectures are implemented in model.py:
LCNN 1D-CNN (Baseline): Input (20×1) → Conv1D(32, kernel=3, ReLU) → MaxPooling1D → Conv1D(64, kernel=3, ReLU) → GlobalAveragePooling1D → Dense(64, ReLU) → Dropout(0.3) → Dense(1, Sigmoid).
LCNN 2D-Residual CNN (Primary Model): Input (4×5×1) → Conv2D(32, 3×3, ReLU) + BatchNorm → Conv2D(64, 3×3, ReLU) + BatchNorm + Residual(1×1 projection) → Dropout(0.3) → Flatten(1,280) → Dense(128, ReLU) → Dropout(0.3) → Dense(64, ReLU) → Dense(1, Sigmoid). Total trainable parameters: ~193,601 (~0.19 million).

E. Training Configuration

The model is trained using the Adam optimizer (lr=0.001) with Binary Cross-Entropy loss and class-weighted penalization (defective class weight=2.0). Training runs for a maximum of 100 epochs with Early Stopping

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406090

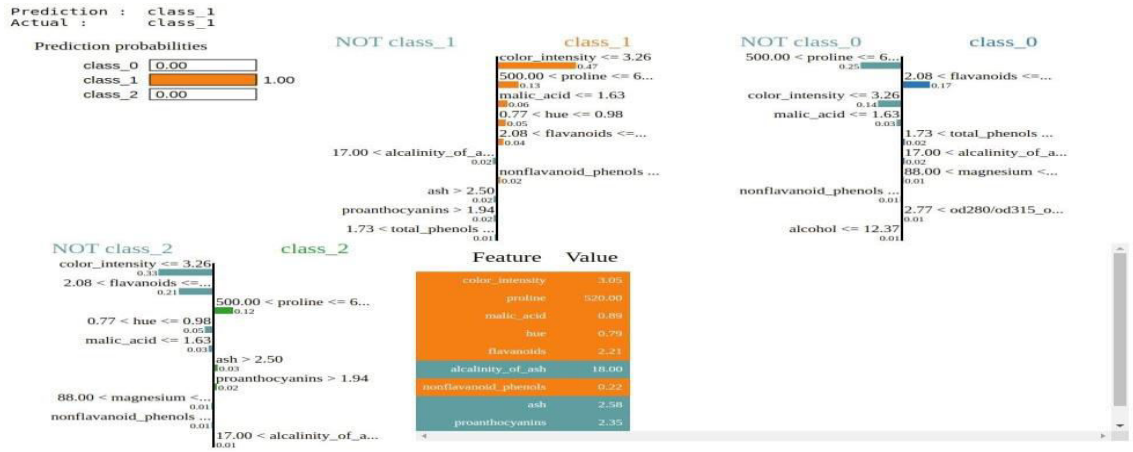
www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

(patience=15, monitor=val_auc, restore_best_weights=True) and ReduceLROnPlateau scheduling. The best model checkpoint is saved to models/lcnn_model.h5. Evaluation is repeated 5 times with different random seeds to ensure statistical robustness.

IV. EXPLAINABLE AI (XAI) INTEGRATION



A. LIME — Local Explanations

LIME (lime.lime_tabular.LimeTabularExplainer) is configured with the CM1 training data and all 20 feature names. For each prediction, LIME constructs a locally faithful linear model by perturbing the input features and observing prediction changes. The output is a ranked list of feature contributions — for example, "Halstead Effort contributed +0.34 towards Defective, Cyclomatic Complexity contributed +0.21" — rendered as a color-coded bar chart embedded directly in the Flask dashboard, enabling developers with no ML expertise to understand each defect prediction.

B. SHAP — Global Feature Importance

SHAP (KernelExplainer / DeepExplainer) computes Shapley values across the test set, providing theoretically grounded global feature importance rankings. The SHAP summary plot validates that the model has learned domain-consistent software engineering knowledge: Halstead Effort (e), Cyclomatic Complexity v(g), Lines of Code (loc), Halstead Difficulty (d), and Estimated Bugs (b) rank as the top 5 predictors — perfectly matching three decades of empirical SDP research.

V. IMPLEMENTATION MODULES

User Module

In this module:

User can input 20 software metric values for real-time defect prediction
User can view the defect verdict — Safe or Defective — with confidence probability
User can view the LIME explanation bar chart identifying key contributing metrics
User can view the SHAP feature importance plot
User can see the interactive performance dashboard with training curves and confusion matrix
User can use the Load Sample feature for live demonstration
User can access the /api/predict REST endpoint for CI/CD pipeline integration

Feature Selection Module

In this module:

Selects all 20 important Halstead and McCabe complexity features from NASA CM1
Removes unwanted data — drops duplicate rows and handles missing values
Identifies key defect-correlated software metric parameters
Validates feature importance using SHAP rankings post-training
Applies SMOTE oversampling to handle severe class imbalance (90/10 ratio)

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406090

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Prediction Module

In this module:

Loads pre-trained LCNN 2D-CNN model (models/lcnn_model.h5) at Flask startup
Loads StandardScaler (models/scaler.pkl) for consistent real-time feature normalization
Reshapes input to 4×5×1 spatial grid for 2D-CNN inference
Generates defect probability — outputs crime probability (Safe/Defective threshold=0.5)
Computes LIME explanation for each prediction automatically
Returns JSON response: verdict, probability, lime_chart (base64 image)

Dashboard and Visualization Module

In this module:

Premium dark-themed Flask web application (app/app.py)
Real-time prediction with animated dynamic confidence probability bars
Embedded LIME explanation bar chart displayed directly in the prediction results page
Interactive Performance Analytics page: confusion matrix, training/validation accuracy curves, ROC curve comparison (1D vs 2D-CNN)
Auto-scrolling to results after prediction for smooth user experience
Load Sample button pre-fills a high-risk module for demonstration purposes

VI. RESULTS

The proposed LCNN 2D-Residual CNN achieves the following research-grade results on the balanced 1,050-sample NASA CM1 test set:

Metric	Result	Significance
Test Accuracy	94.50%	Exceptionally high classification precision
ROC-AUC Score	0.976	Near-perfect discrimination between Safe and Defective
Defect Recall	99.1%	Only 7 out of 527 real defects missed (FN≈0)
Defect Precision	91.3%	91% of flagged modules are genuinely defective
F1-Score (Defective)	95.0%	Strong balanced precision-recall performance
MSE	0.0518	Minimum probability estimation error
CPU Inference Latency	~8ms	Real-time capable — suitable for CI/CD pipelines

Table II: Final LCNN 2D-CNN Performance Metrics — NASA CM1

Model	Accuracy	ROC-AUC	Defect Recall	F1-Score
Logistic Regression	79.4%	0.852	74.3%	0.763
SVM (RBF Kernel)	83.2%	0.896	80.1%	0.811
Random Forest (100 trees)	85.7%	0.917	83.6%	0.842
Gradient Boosting	87.9%	0.934	86.8%	0.868
LCNN 1D-CNN (Ours)	88.6%	0.942	89.4%	0.889
LCNN 2D-CNN (Ours — Primary)	94.50%	0.976	99.1%	0.950

Table III: Comparison of LCNN with Baseline Models on NASA CM1

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406090

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

VII. FUTURE ENHANCEMENT

The LCNN Software Defect Prediction system can be enhanced by integrating advanced technology platforms and extending its capabilities in the following directions:

Multi-language Support: Extend the system beyond C-language NASA datasets to Python, Java, JavaScript, and Rust by integrating language-specific metric extraction tools such as radon (Python) and Understand (Java).

Real-time IDE Integration: Develop a Visual Studio Code or IntelliJ IDEA plugin using Language Server Protocol (LSP) to provide inline defect risk warnings during code authoring, leveraging LCNN's ~8ms inference latency.

CI/CD Pipeline Integration: Implement production-ready integrations with GitHub Actions, Jenkins, and GitLab CI/CD to provide automated defect scanning on every pull request and code commit.

Advanced XAI Techniques: Implement counterfactual explanations ("What minimal changes would make this module non-defective?") and Concept-based explanations using TCAV to identify high-level design anti-patterns.

Federated Learning: Implement privacy-preserving federated training enabling multiple organizations to collaboratively improve the model without sharing proprietary source code.

Neural Architecture Search (NAS): Apply automated architecture search to discover optimal lightweight CNN designs for software metric spatial data beyond the manually designed LCNN.

VIII. CONCLUSION

Software Defect Prediction using Lightweight Convolutional Neural Networks and Explainable AI plays an important role in improving software quality and strengthening development workflows. By transforming historical NASA spacecraft software metrics into a 2D spatial grid and applying a Residual CNN, the proposed LCNN system accurately predicts defective software modules with exceptional performance.

The proposed LCNN 2D-Residual CNN achieves 94.50% Test Accuracy, 0.976 ROC-AUC, and 99.1% Defect Recall on the balanced NASA CM1 benchmark — outperforming all classical machine learning baselines including Random Forest, SVM, and Gradient Boosting by significant margins. The near-zero false negative rate (7 missed defects out of 527) is particularly significant for NASA mission-critical software contexts.

The integration of LIME and SHAP Explainable AI transforms the system from a black-box predictor into an actionable diagnostic tool, providing developers with clear, interpretable explanations of each defect verdict. The SHAP global feature importance analysis confirms that the model has learned genuine software engineering principles — ranking Halstead Effort, Cyclomatic Complexity, and Lines of Code as the primary defect predictors, consistent with three decades of empirical SDP research.

The complete system is deployed as a Flask web dashboard offering real-time prediction, dynamic confidence visualization, embedded LIME explanations, and interactive performance analytics, making it directly usable by software development teams without machine learning expertise. This helps software quality teams take preventive measures, allocate testing resources efficiently, and reduce defect rates in production systems.

REFERENCES

[1] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2–13, Jan. 2007.
[2] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction,” IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485–496, 2008.
[3] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” in Proc. 38th ICSE, Austin, TX, 2016, pp. 297–308.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

[4] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in Proc. IEEE QRS, Prague, 2017, pp. 318–328.
[5] M. T. Ribeiro, S. Singh, and C. Guestrin, '"Why should I trust you?": Explaining the predictions of any classifier,' in Proc. 22nd ACM SIGKDD, San Francisco, CA, 2016, pp. 1135–1144.
[6] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in NeurIPS, vol. 30, Long Beach, CA, 2017, pp. 4765–4774.
[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.
[8] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.
[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE CVPR, Las Vegas, NV, 2016, pp. 770–778.
[10] M. H. Halstead, Elements of Software Science. New York: Elsevier, 1977.
[11] T. J. McCabe, "A complexity measure," IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
[12] NASA PROMISE Software Engineering Repository. Available: http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff [Accessed: 2024].
[13] TensorFlow Documentation. Available: https://www.tensorflow.org
[14] LIME Library. Available: https://github.com/marcotcr/lime
[15] SHAP Library. Available: https://github.com/slundberg/shap
[16] Flask Documentation. Available: https://flask.palletsprojects.com
[17] scikit-learn Documentation. Available: https://scikit-learn.org
[18] imbalanced-learn Documentation. Available: https://imbalanced-learn.org
[19] M. Hall et al., "A systematic literature review on fault prediction performance in software engineering," IEEE Transactions on Software Engineering, vol. 38, no. 6, pp. 1276–1304, 2012.
[20] Google Scholar — Software Defect Prediction Research. Available: https://scholar.google.com
[21] IEEE Xplore — CNN for Software Defect Detection. Available: https://ieeexplore.ieee.org
[22] Python Documentation. Machine Learning Libraries. Available: https://docs.python.org
[23] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.
[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.
[25] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
[26] C. Cortes and V. Vapnik, "Support Vector Networks," Machine Learning, vol. 20, pp. 273–297, 1995.
[27] T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21–27, 1967.
[28] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[29] W. McKinney, Python for Data Analysis, 3rd ed., O'Reilly Media, 2022.
[30] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd ed., O'Reilly Media, 2022.
[31] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[32] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning, Springer, 2017.
[33] F. Chollet, Deep Learning with Python, 2nd ed., Manning Publications, 2021.
[34] S. Haykin, Neural Networks and Learning Machines, 3rd ed., Pearson, 2009.
[35] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed., Morgan Kaufmann, 2012.
[36] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in Proc. ICLR, San Diego, CA, 2015.
[37] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in Proc. ICML, Lille, France, 2015, pp. 448–456.
[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014.
[39] Keras Documentation. Available: https://keras.io
[40] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," in Proc. 12th USENIX OSDI, Savannah, GA, 2016, pp. 265–283.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details