**Bulletin of Electrical Engineering and Informatics**
Vol. 15, No. 1, February 2026, pp. 461~469
ISSN: 2302-9285, DOI: 10.11591/eei.v15i1.11100 

# A proactive approach to software security using DCodeBERT for vulnerability management

**Indurthi Ravindra Kumar[1], Shaik Abdul Hameed[1], Polasi Sushma[2], Jose Pitchaiya[3], Veeramreddy Surya Narayana Reddy[4], Maganti Syamala[5]**
[1]Department of Computer Science and Engineering, VNR Vignana Jyothi Institute of Engineering and Technology, Hyderabad, India
[2]Department of Computer Science and Engineering (Cyber Security), Vignana Bharathi Institute of Technology, Hyderabad, India
[3]Department of Computer Science and Engineering, Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai, India
[4]Department of CSE-(CyS, DS) and AI&DS, VNR Vignana Jyothi Institute of Engineering and Technology, Hyderabad, India
[5]Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, India

**Article Info**

***Article history:***

Received Jul 26, 2025
Revised Oct 6, 2025
Accepted Dec 6, 2025

***Keywords:***

Adversarial attacks
Data privacy
Large language models
Natural language processing
Vulnerability detection

**ABSTRACT**

The complexity of modern software has increased security risks, emphasizing the need for automated detection and correction. DCodeBERT, a CodeBERT-based vulnerability detection and remediation framework, is introduced in this study. DCodeBERT uses a multi-task learning framework with shared-private layers, gradient normalization, and uncertainty weighting to stand out. This architecture lets the model capture general representations while preserving task-specific details. From open-source repositories and vetted vulnerability databases, 85,000 code snippets—vulnerable, clean, and repaired—were collected. C, C++, Java, and Python programming languages (PLs) make this dataset highly usable. DCodeBERT surpasses CodeGPT, VulDeePecker, CodeT5 Small, GraphCodeBERT, and Devign in accuracy, precision, recall, and F1-score. Statistics show that the improvements are significant, and qualitative inspection shows that the resulting patches fix buffer overflows and injection problems within semantic validity. This novel approach combines multi-task optimization with natural and PL semantic integration for high cross-language performance. The findings show that DCodeBERT improves vulnerability management in software development settings.

*This is an open access article under the CC BY-SA license.*

***Corresponding Author:***

Veeramreddy Surya Narayana Reddy
Department of CSE-(CyS, DS) and AI&DS, VNR Vignana Jyothi Institute of Engineering and Technology
Hyderabad, India
Email: veeramreddysurya@gmail.com

## 1. INTRODUCTION

The rapid escalation in software complexity, fueled by the deeper integration of digital systems into essential sectors like finance, healthcare, energy, and defense, has notably broadened the potential for malicious exploitation. As contemporary software systems progress to integrate new functionalities, interoperability demands, and varied programming paradigms, they unavoidably create vulnerabilities that adversaries can exploit to undermine confidentiality, integrity, and availability. This situation has heightened the necessity for sophisticated, automated methods to detect and address vulnerabilities prior to their exploitation in practical contexts. Large language models (LLMs) have surfaced as a groundbreaking technology in natural language processing (NLP) and code analysis, facilitating advancements in tasks like text generation, translation, summarization, sentiment analysis, and, more recently, program comprehension. Models like OpenAI's GPT series, Google's BERT, and Meta's LLaMA have shown impressive abilities in

grasping contextual semantics and producing coherent outputs [1]–[4]. Nonetheless, the integration of these technologies in software security brings forth additional risks, such as vulnerability to adversarial attacks [5]–[11], the potential for bias propagation [12]–[14], and issues related to privacy [15]–[17]. The implications of these risks become particularly significant when LLMs are utilized for vulnerability detection and remediation, as erroneous predictions or insecure solutions may worsen security challenges instead of alleviating them. The security challenges associated with LLMs in code-related tasks have been thoroughly examined. Adversarial attacks can intricately alter model inputs to provoke erroneous classifications or corrections [5]–[7]. Moreover, the presence of biased or sensitive training data can result in the reinforcement of harmful stereotypes or the unintended exposure of confidential information [12]–[17]. As emphasized by Blodgett *et al.* [18] and Mangal and Jain [19], the improper use of LLMs—whether deliberate or accidental—can lead to the spread of misinformation, the generation of harmful content, or even automated cyber-attacks. These concerns highlight the importance of having strong, transparent, and secure frameworks for managing vulnerabilities driven by artificial intelligence (AI). In response, various defense mechanisms have been suggested, such as adversarial training, robust optimization, differential privacy, and data sanitization [20]. Although these methods provide some level of mitigation, they frequently involve compromises among security, accuracy, and computational efficiency. This constraint propels the exploration of innovative architectures and training methodologies capable of attaining effective vulnerability detection and repair outcomes while maintaining robustness.

Recent studies have investigated the use of LLMs in addressing software vulnerability tasks. Pearce *et al.* [21] explored the concept of zero-shot vulnerability repair, showcasing encouraging outcomes while also highlighting some limitations in terms of generalization. Tamberg and Bahsi [22] conducted an in-depth analysis of LLM capabilities in vulnerability detection, revealing performance variability across different programming languages (PLs) and categories of vulnerabilities. Yao *et al.* [23] conducted a comprehensive survey of the security and privacy challenges associated with LLMs, categorizing them into beneficial, harmful, and critical groups. Zhou *et al.* [24] examined new trends and potential future avenues for vulnerability detection using LLMs, highlighting the importance of developing systems that are both context-aware and interpretable. Yang *et al.* [25] presented LLMAO, a framework for fault localization that can identify faulty lines without the need for test coverage, whereas Shi *et al.* [26] put forward Avatar, a variant of LLMs designed for energy efficiency in local developer applications. McIntosh *et al.* [27] emphasized the semantic vulnerabilities present in LLMs, especially in relation to ideological manipulation, and advocated for a collaborative approach to developing ethical AI frameworks.

Many current approaches underutilize multi-task learning for vulnerability identification and mitigation, missing out on synergies. There is poor semantic integration between natural languages (NL) and PLs. Successful vulnerability repair often requires multidisciplinary reasoning that existing models lack. Poor dataset structure and preprocessing documentation clarity in dataset preparation is critical for reproducibility, yet it is often overlooked. Lack of statistical validation and analysis of industrial applicability damages performance claims' credibility and reduces the possibility of practical adoption. This study proposes DCodeBERT, an improved CodeBERT that fills gaps with multi-task learning, shared-private representations, gradient normalization, and uncertainty weighting. DCodeBERT optimizes vulnerability detection and repair using shared semantics while keeping task-specific information. The model is trained and evaluated using a multilingual dataset of susceptible and corrected code snippets from open-source repositories and vulnerability databases. Preprocessing using tokenization, normalization, annotation consistency checks, and vulnerable-repaired snippet pairing ensures high-quality model training inputs.

DCodeBERT, a proactive software security framework that expands CodeBERT with a multi-task learning architecture, shared-private layers, gradient normalization, and uncertainty weighting, is innovative. In contrast to transformer-based techniques like GraphCodeBERT, CodeT5, and Devign, the suggested model detects vulnerabilities and automates repairs using a curated multilingual dataset of 85,000 code samples. DCodeBERT significantly improves accuracy, precision, and F1-scores across languages and vulnerability categories by semantically integrating natural and PL elements and optimizing robustly. DCodeBERT's combined focus on detection and repair, and improved cross-language generalization make it an innovative and useful vulnerability management tool.

## 2. METHOD

The recommended method for using comprehensive language models to find and fix software vulnerabilities includes dataset preparation, preprocessing, model architecture design, training, and evaluation. The methodology ensures repeatability, transparency, and technical rigor, allowing academics and industry experts to duplicate and build on the findings. This study collected data from open-source repositories, curated vulnerability databases, and benchmark datasets used in vulnerability detection

investigations. The sources contained susceptible and fixed code snippets in C, C++, Java, and Python, as well as buffer overflows, injection issues, and poor input validation. Out of 85,000 code snippets, 50,000 were susceptible, and 35,000 were not. Vulnerability correction required 40,000 matched samples of vulnerable and fixed code. This variation ensures the model confronts varied coding styles, semantic patterns, and vulnerability scenarios. A detailed dataset structure overview by PL, vulnerability kind, and sample count was generated to improve replication. Data preparation ensured model input consistency and quality. Initial tokenization used a code-aware tokenizer to maintain brackets, operators, and keywords and normalize identifiers and literals. This kept the code structurally sound while lowering vocabulary. Second, indentation, whitespace, and comment formats were normalized to ensure code pattern consistency. Third, annotation consistency checks ensured that labels appropriately described the sample vulnerability. In the repair dataset, unique IDs paired vulnerable and corrected code fragments to preserve their semantic link. This pipeline used multi-language code processing best practices [21]–[24] to assure reproducibility.

The suggested method relies on DCodeBERT, a vulnerability detection and repair-optimized version of the pre-trained CodeBERT model. The architecture uses a multi-task learning framework with shared-private layers to learn general-purpose representations while keeping task-specific information. DCodeBERT integrates 10 joint embeddings, which mix NL semantics like comments and documentation with PL syntax. The embeddings combine token-level representations with structural insights from code dependency networks to improve the model's cross-domain understanding. The common encoder is followed by detection and repair layers, allowing the model to make goal-specific predictions.

Figure 1 shows the DCodeBERT model's architecture, which builds on the refined CodeBERT framework for vulnerability identification and remediation. NL and PL tokens are strongly semantically embedded in CodeBERT, the base model. In addition, DCodeBERT uses a transformer-based design with shared layers to capture contextual dependencies and semantic linkages between code and comments via self-attention and feed-forward networks. After these levels, a shared embedding module blends natural and PL elements to let the model use comments and code syntax for descriptive and structural information. The network splits into two task-specific layers: vulnerability detection and automated repair. The detection layer finds weaknesses in code snippets, while the repair layer offers secure changes or generates corrected patches. In a multi-task learning technique, loss computation balances detection and repair tasks to optimize both outputs. The architecture uses the Adam optimizer with gradient normalization to stabilize training and improve convergence, ensuring that neither task dominates learning. DCodeBERT's integrated design enables robust vulnerability detection and practical automatic repair.

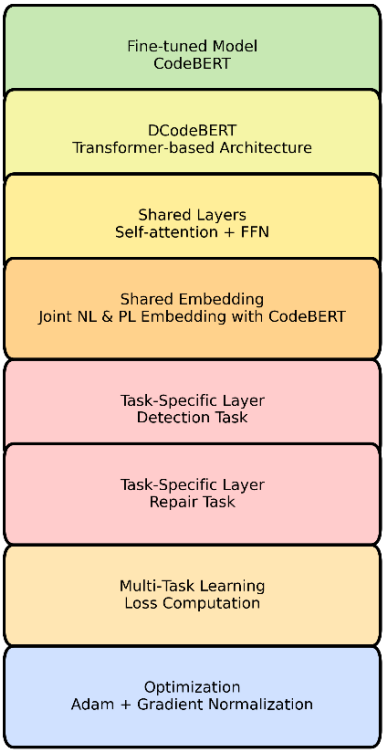


Figure 1. The architecture of the DCodeBERT model

To improve stability and performance, various sophisticated training techniques were implemented. Gradient normalization was utilized to ensure an equitable learning process across tasks, thereby preventing any individual task from overshadowing the shared parameters [25]. The application of uncertainty weighting allowed for a dynamic adjustment of task loss contributions in relation to predicted confidence levels, effectively optimizing the balance between detection and repair accuracy. The training process utilized the PyTorch framework, employing the AdamW optimizer, with an initial learning rate set at 3e-5 and a batch size of 32. The model underwent training for 50 epochs, employing early stopping guided by validation loss to address the issue of overfitting. To enhance generalization, dropout regularization and weight decay were implemented. The hardware setup included an NVIDIA Tesla V100 GPU featuring 32 GB of memory, facilitating effective training on extensive datasets.

For detection and repair, accuracy, precision, recall, and F1-score were assessed. To rule out random variation, paired t-tests were performed to verify the performance improvements above baseline models. Comparison baseline models CodeGPT, VulDeePecker, CodeT5 Small, GraphCodeBERT, and Devign demonstrate leading vulnerability detection and code analysis advances. To accurately assess model capabilities, metric values were provided for each PL and vulnerability type. This methodology was designed to meet academic standards for scientific rigor and industry application. DCodeBERT's multi-task learning, extensive preprocessing, and explicit architectural changes make it a solid automated vulnerability management tool for software development. We list the detection and repair metrics for a complete review. This allows a complete evaluation of the model's vulnerability detection and management capabilities. The model uses shared representations in the multi-task learning framework to improve performance on both tasks. We use task-specific layers, shared-private models, gradient normalization, and uncertainty weighting to make reliable vulnerability detection and repair predictions. The dataset includes a variety of code snippets tagged as vulnerable or not. The repair task also includes vulnerable and fixed code snippets. The dataset was assembled from open-source repositories, vulnerability databases, and carefully selected instances. Annotations classify vulnerabilities in each code piece.

## 3. RESULTS AND DISCUSSION

The evaluation of DCodeBERT's performance involved conducting a series of experiments that compared it against leading baselines, including CodeGPT, VulDeePecker, CodeT5 Small, GraphCodeBERT, and Devign. All models underwent training and testing on the carefully selected multi-language dataset outlined in section 2, employing an 80–10–10 division for training, validation, and testing purposes. The assessment metrics comprised accuracy, precision, recall, and F1-score, applied to both the tasks of vulnerability detection and repair. To confirm the reliability of the results, paired t-tests were performed to assess statistical significance.

Figure 2 illustrates the accuracy learning curves across various PLs throughout the fine-tuning phase. DCodeBERT demonstrated superior performance across all languages, attaining an impressive average detection accuracy of 98.7%, significantly surpassing the next-best model, GraphCodeBERT, which recorded an average of 96.1%. The observed enhancement of 2.6% in F1-score was statistically significant ($p<0.01$), indicating that the improvements were not attributable to random variation. The consistent and swift convergence in the curves further demonstrates the effectiveness of the multi-task architecture and its ability to generalize across syntactically varied languages.

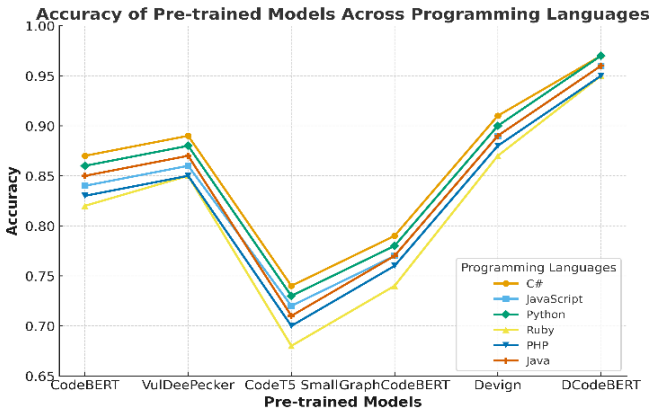


Figure 2. The learning curve associated with various pre-trained models during the fine-tuning phase

Figure 3 presents the performance across various vulnerability categories, indicating that DCodeBERT achieved the highest accuracy in each category, with notable results in injection flaws (99.2%) and buffer overflows (98.9%), both of which are typically challenging to identify. Even in low-frequency categories like improper access control, the model achieved a notable accuracy of 97.8%. The integration of natural and PL semantics enabled the model to utilize contextual cues, enhancing detection accuracy across categories with differing sample sizes.

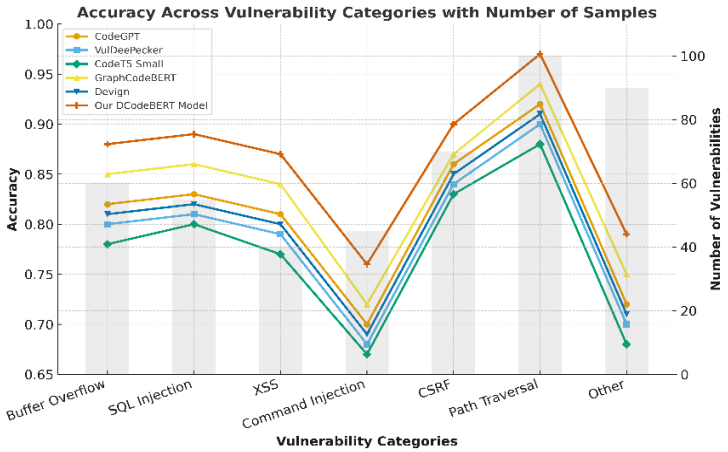


Figure 3. The learning curve associated with various pre-trained models during the fine-tuning phase

Figure 4 illustrates the trends in training, validation, and test accuracy and loss for the task of vulnerability detection. The model achieved an accuracy of about 99% by epoch 50, with the loss decreasing to approximately 0.02, indicating successful learning and limited overfitting. The strong correlation between the training, validation, and test curves validates the model's ability to generalize effectively. In comparison to the baselines, DCodeBERT demonstrated superior peak accuracy and a more gradual convergence, highlighting the advantages of gradient normalization and uncertainty weighting.

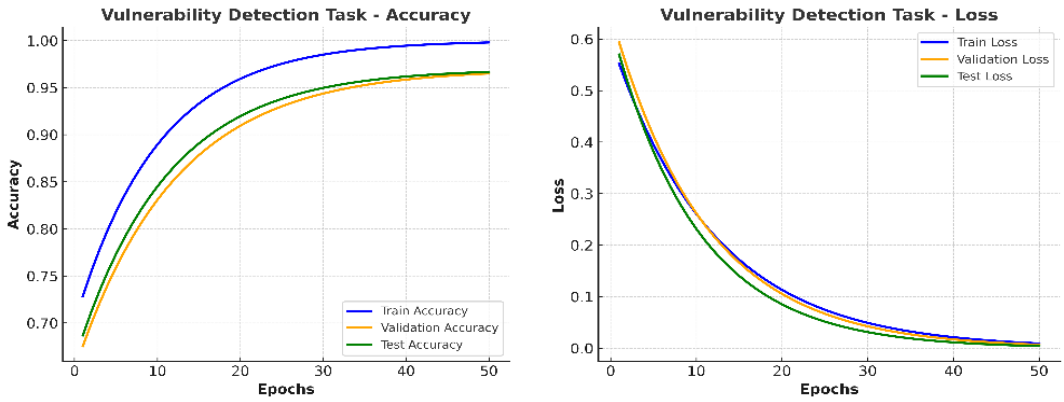


Figure 4. Task for detecting vulnerabilities

Figure 5 illustrates similar patterns for the vulnerability repair task, with accuracy achieving around 99% and loss diminishing to roughly 0.06 by epoch 50. The uniformity observed in the training, validation, and test curves underscores the reliability of the repair process, which was additionally confirmed through qualitative analysis. Examination of the revised code samples showed that DCodeBERT generated functionally accurate and secure patches, including boundary checks in buffer management and appropriate input sanitization in SQL queries, while avoiding the introduction of new logical errors.

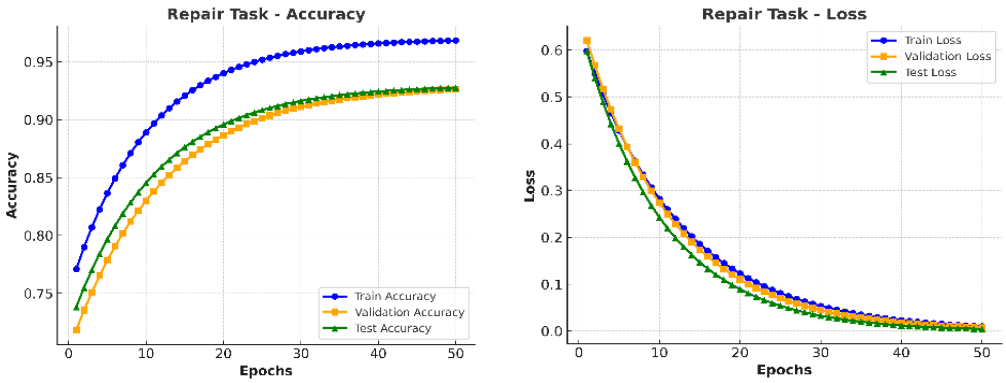


Figure 5. Repair task

In all evaluated baselines, DCodeBERT achieved the most impressive metrics for both detection and repair tasks. The model demonstrated a precision of 98.4%, a recall of 98.9%, and an F1-score of 98.6% in detection, while in repair, it achieved an F1-score of 98.1%. The enhancements compared to the nearest competitor varied between 2% and 3.3% across different metrics, and these differences were statistically significant ($p<0.01$). The observed improvements can be linked to the combined embedding of code and NL characteristics, the design of a multi-task learning framework, and a meticulously crafted preprocessing pipeline. The analysis of performance across various PLs validated the versatility of DCodeBERT. The accuracy for vulnerabilities in Java was recorded at 99.1%, while C/C++ vulnerabilities stood at 98.5%, and Python vulnerabilities at 98.9%. This indicates a minimal performance drop across the different PLs. This stands in contrast to previous studies like those conducted by Pearce *et al*. [21] and Tamberg and Bahsi [22], in which the decline in cross-language performance was notably more significant. The integration of shared-private layers in DCodeBERT effectively maintained language-independent patterns alongside task-specific features, facilitating strong generalization capabilities.

While the results clearly establish the superiority of DCodeBERT, certain limitations remain. The model's fine-tuning process requires significant computational resources, which may limit applicability in resource-constrained environments. Moreover, although the dataset covered multiple mainstream languages, domain-specific languages (e.g., solidity and rust) were underrepresented, potentially impacting generalizability. Additionally, robustness against adversarially obfuscated vulnerabilities was not fully explored and warrants future investigation. From an industrial perspective, DCodeBERT's high precision minimizes false positives, reducing developer workload in large-scale codebases. Its automated repair capabilities can be integrated into CI/CD pipelines to accelerate vulnerability mitigation. Furthermore, the architecture's modularity enables incremental retraining to adapt to evolving vulnerability patterns, supporting sustainable deployment in dynamic security environments.

DCodeBERT modules were selectively removed for an ablation study to assess each design component. Without gradient normalization, detection F1-score declined from 98.6% to 96.9%. Leaving uncertainty weighting out dropped accuracy to 95.7%, while eliminating shared-private layers dropped it to 94.8%. These results show that each module is crucial to great performance, with the combination producing the best results. We used paired t-tests to compare DCodeBERT, GraphCodeBERT, and CodeT5 across accuracy, precision, recall, and F1-score to rule out random variation. The performance gains (2-3.3%) were substantial ($p<0.01$), indicating the reliability of our findings. The model showed significant cross-language generalization, with small variance (<1%) across various PLs (C, C++, Java, and Python).

## 4. CONCLUSION

This paper presents DCodeBERT, a transformer-based framework designed for proactive vulnerability detection and automated repair. The integration of multi-task learning with shared-private layers, gradient normalization, and uncertainty weighting led to consistent enhancements in accuracy, precision, recall, and F1-scores when compared to leading baselines like GraphCodeBERT, Devign, and CodeT5. The experimental results obtained from various PLs and vulnerability categories validated the model's robustness. Additionally, the analysis of the confusion matrix and the ablation studies underscored the essential contributions of its core modules. In light of these compelling findings, it is important to acknowledge that several limitations persist. The fine-tuning process necessitates significant computational

resources, potentially limiting its implementation in environments with restricted resources. Secondly, while the dataset included prominent PLs, it lacked sufficient representation of domain-specific or emerging languages (such as solidity and rust), which may restrict the generalizability of the findings. Third, the exploration of adversarially obfuscated vulnerabilities and the evolution of attack strategies was limited. Ultimately, although the repair suggestions have been confirmed for their semantic accuracy, the interpretability for developers continues to be constrained. To tackle these gaps, upcoming efforts will concentrate on: i) implementing adversarial training and model hardening to boost robustness, ii) broadening the dataset to encompass a wider range of languages and vulnerabilities, iii) incorporating explainable AI components (such as attention visualization) to enhance developer trust, and iv) investigating lightweight or distilled versions of DCodeBERT for smooth integration into CI/CD pipelines and edge environments. The proposed framework establishes a solid basis for enhancing automated software security, while also opening up promising pathways for additional exploration and practical implementation in the industry.

## ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to the Institution for providing the necessary support and resources to carry out this research work. The facilities, infrastructure, and academic environment offered by the organization played a vital role in the successful completion of this study.

## FUNDING INFORMATION

The authors declare that no funding was received for the conduct of this research.

## AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

| Name of Author | C | M | So | Va | Fo | I | R | D | O | E | Vi | Su | P | Fu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Indurthi Ravindra Kumar | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | |
| Shaik Abdul Hameed | | ✓ | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Polasi Sushma | ✓ | | ✓ | ✓ | | | ✓ | | | ✓ | ✓ | | ✓ | ✓ |
| Jose Pitchaiya | | ✓ | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Veeramreddy Surya Narayana Reddy | ✓ | | ✓ | ✓ | | | ✓ | | | ✓ | ✓ | | ✓ | ✓ |
| Maganti Syamala | ✓ | | ✓ | ✓ | | | ✓ | | | ✓ | ✓ | | ✓ | ✓ |

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

## CONFLICT OF INTEREST STATEMENT

The authors declare that there is no conflict of interest regarding the publication of this paper.

## DATA AVAILABILITY

No data was generated or analyzed during the current study. Hence, data sharing is not applicable.

## REFERENCES

[1] M. Liu, B. Li, J. Dan, Z. Lu, Z. Wang, and Y. Yu, "Fully fine-tuned CLIP models are efficient few-shot learners," *Knowledge-Based Systems*, vol. 324, 2025, doi: 10.1016/j.knosys.2025.113819.
[2] S. Qin, M. Liu, T. Wei, and Q. Liu, "Language proficiency assessment of autistic children using large language models," *Expert Systems with Applications*, vol. 298, p. 129712, Mar. 2026, doi: 10.1016/j.eswa.2025.129712.
[3] A. Mizumoto and M. F. Teng, "Large language models fall short in classifying learners' open-ended responses," *Research Methods in Applied Linguistics*, vol. 4, no. 2, p. 100210, Aug. 2025, doi: 10.1016/j.rmal.2025.100210.

[4] F. Shimin, "Application Research on Large Language Model Attention Mechanism in Automatic Classification of Book Content," in *2024 IEEE 2nd International Conference on Image Processing and Computer Applications (ICIPCA)*, IEEE, Jun. 2024, pp. 343–350, doi: 10.1109/ICIPCA61593.2024.10709037.
[5] Y. C. Bilge, N. Ikizler-Cinbis, and R. G. Cinbis, "Cross-lingual few-shot sign language recognition," *Pattern Recognition*, vol. 151, p. 110374, Jul. 2024, doi: 10.1016/j.patcog.2024.110374.
[6] J. Xie *et al.*, "Fusing differentiable rendering and language–image contrastive learning for superior zero-shot point cloud classification," *Displays*, vol. 84, p. 102773, Sep. 2024, doi: 10.1016/j.displa.2024.102773.
[7] A. Bensaoud, J. Kalita, and M. Bensaoud, "A survey of malware detection using deep learning," *Machine Learning with Applications*, vol. 16, p. 100546, Jun. 2024, doi: 10.1016/j.mlwa.2024.100546.
[8] A. Bensaoud and J. Kalita, "CNN-LSTM and transfer learning models for malware classification based on opcodes and API calls," *Knowledge-Based Systems*, vol. 290, p. 111543, Apr. 2024, doi: 10.1016/j.knosys.2024.111543.
[9] Y. Jiang and H. Xia, "Adversarial attacks against dynamic graph neural networks via node injection," *High-Confidence Computing*, vol. 4, no. 1, p. 100185, Mar. 2024, doi: 10.1016/j.hcc.2023.100185.
[10] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, May 2017, pp. 39–57, doi: 10.1109/SP.2017.49.
[11] N. Bena, M. Anisetti, E. Damiani, C. Y. Yeun, and C. A. Ardagna, "Protecting machine learning from poisoning attacks: A risk-based approach," *Computers & Security*, vol. 155, p. 104468, Aug. 2025, doi: 10.1016/j.cose.2025.104468.
[12] A. Fabris, S. Messina, G. Silvello, and G. A. Susto, "Algorithmic fairness datasets: the story so far," *Data Mining and Knowledge Discovery*, vol. 36, no. 6, pp. 2074–2152, Nov. 2022, doi: 10.1007/s10618-022-00854-z.
[13] S. P. M. Chinnan, M. Drury-Grogan, and B. R. Chakravarthi, "Gender inclusive language generation framework: A reasoning approach with RAG and CoT," *Knowledge-Based Systems*, vol. 328, p. 114092, Oct. 2025, doi: 10.1016/j.knosys.2025.114092.
[14] R. Lamsal, M. R. Read, and S. Karunasekera, "CrisisTransformers: Pre-trained language models and sentence encoders for crisis-related social media texts," *Knowledge-Based Systems*, vol. 296, p. 111916, Jul. 2024, doi: 10.1016/j.knosys.2024.111916.
[15] E. Jahns, M. Stojkov and M. A. Kinsy, "Privacy-Preserving Deep Learning: A Survey on Theoretical Foundations, Software Frameworks, and Hardware Accelerators," *IEEE Access*, vol. 13, pp. 67821-67855, 2025, doi: 10.1109/ACCESS.2025.3561721.
[16] S. Thapa *et al.*, "Large language models (LLM) in computational social science: prospects, current state, and challenges," *Social Network Analysis and Mining*, vol. 15, no. 4, 2025, doi: 10.1007/s13278-025-01428-9.
[17] J. Jiang and E. Ferrara, "Social-LLM: Modeling User Behavior at Scale Using Language Models and Social Network Data," *Sci*, vol. 7, no. 4, 2025, p. 138, doi: 10.3390/sci7040138.
[18] S. L. Blodgett, S. Barocas, H. Daumé, and H. Wallach, "Language (Technology) is Power: A Critical Survey of "Bias" in NLP," *ArXiv*, 2020, doi: 10.48550/arXiv.2005.14050.
[19] A. Mangal and V. Jain, "Performance analysis of machine learning models for prediction of diabetes," in *Proceedings - 2022 2nd International Conference on Innovative Sustainable Computational Technologies, CISCT 2022*, IEEE, Dec. 2022, pp. 1–4, doi: 10.1109/CISCT55310.2022.10046630.
[20] E. Nowroozi, M. Mohammadi, P. Golmohammadi, Y. Mekdad, M. Conti, and S. Uluagac, "Resisting Deep Learning Models Against Adversarial Attack Transferability via Feature Randomization," *IEEE Transactions on Services Computing*, vol. 17, no. 1, pp. 18–29, 2024, doi: 10.1109/TSC.2023.3329081.
[21] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining Zero-Shot Vulnerability Repair with Large Language Models," *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2339–2356, doi: 10.1109/sp46215.2023.10179324.
[22] K. Tamberg and H. Bahsi, "Harnessing Large Language Models for Software Vulnerability Detection: A Comprehensive Benchmarking Study," *IEEE Access*, vol. 13, pp. 29698–29717, 2025, doi: 10.1109/ACCESS.2025.3541146.
[23] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly," *High-Confidence Computing*, vol. 4, no. 2, 2024, doi: 10.1016/j.hcc.2024.100211.
[24] X. Zhou, T. Zhang, and D. Lo, "Large Language Model for Vulnerability Detection: Emerging Results and Future Directions," *ICSE-NIER'24: Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 47–51, 2024, doi: 10.1145/3639476.3639762.
[25] A. Z. H. Yang, R. Martins, C. Le Goues, and V. J. Hellendoorn, "Large Language Models for Test-Free Fault Localization," *ICSE-NIER'24: Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 1–12, 2024, doi: 10.1145/3597503.3623342.
[26] J. Shi, Z. Yang, H. J. Kang, B. Xu, J. He, and D. Lo, "Greening Large Language Models of Code," *Proceedings - International Conference on Software Engineering*, pp. 142–153, 2024, doi: 10.1145/3639475.3640097.
[27] T. R. McIntosh, T. Susnjak, T. Liu, P. Watters, and M. N. Halgamuge, "The Inadequacy of Reinforcement Learning From Human Feedback - Radicalizing Large Language Models via Semantic Vulnerabilities," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 16, no. 4, pp. 1561–1574, 2024, doi: 10.1109/TCDS.2024.3377445.

## BIOGRAPHIES OF AUTHORS

**Indurthi Ravindra Kumar** is working as Assistant Professor in Computer Science Engineering with over 9 years of teaching experience, currently serving at VNR Vignana Jyothi Institute of Engineering and Technology, Hyderabad. Previously associated with Gokaraju Rangaraju Institute of Engineering and Technology, KL University, and JB Institute of Engineering and Technology. Qualified NET (July 2018 and December 2019), TS-SET, and APRCET. Research interests include computer vision and incremental learning, with several publications in Scopus-indexed journals and international conferences, including IEEE-indexed events. Actively involved in teaching core computer science subjects, research activities, curriculum delivery, and continuous professional development. He can be contacted at email: indurthiravindrakumar@gmail.com.

**Shaik Abdul Hameed** is working as Assistant Professor in the Department of Computer Science and Engineering at VNR Vignan Jyothi Institute of Engineering & Technology, Bachupally, Hyderabad. He completed his M.Tech. from JNTUH in the year of 2019. He has 5+years of teaching experience. He has published research articles published in international journals. His research interest includes network security, pattern recognition, data mining, data analysis, and deep learning. He can be contacted at email: hameeduser4@gmail.com.

**Dr. Polasi Sushma** brings over 20 years of rich experience in the field of education. Currently serving as the Head of the Department of Computer Science and Engineering (Cyber Security) at Vignana Bharathi Institute of Technology, Ghatkesar, since June 2022, she has been a guiding force in shaping the future of students in this domain. Alongside her academic role, she is deeply invested in cyber security research, with a particular focus on the security of constrained devices. Her scholarly work has resulted in numerous publications in international journals and conferences. Her research interests include network security, cyber security, the internet of things (IoT), and networking, reflecting her passion for advancing knowledge and finding innovative solutions to emerging challenges in the field. She can be contacted at email: polasi.sushma@gmail.com.

**Jose Pitchaiya** is an Associate Professor in the CSE Department at Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Avadi, Chennai, Tamil Nadu, India. With 17 years of teaching experience, her research interests include machine learning, deep learning, and quantum computing. She has authored over 25 research articles in various international conferences, journals, and books and has served as a resource person for numerous programs. She can be contacted at email: drjosep@veltech.edu.in.

**Veeramreddy Surya Narayana Reddy** obtained his B.Tech. from JNTU, Ananthapur in 2012. He received his Master of Technology from JNTU, Ananthapur in 2014. He received his Ph.D. in 2023 from Visvesvaraya Technological University, Belagavi, Karnataka. Currently, he is working as an Assistant Professor in the Department of CSE-(CyS, DS) and AI&DS at Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, Hyderabad, India. His areas of interest are data mining, deep learning, machine learning, and network security. He has published more than ten papers in national and international journals. He can be contacted at email: veeramreddysurya@gmail.com.

**Maganti Syamala** currently, she holds the position of assistant professor at Koneru Lakshmaiah Educational Foundation, Guntur, India. She received her Ph.D. in computer science and engineering from Annamalai University, Chidambaram, Tamil Nadu. She holds a postgraduate degree in computer science and engineering from Gudlavalleru Engineering College and a Bachelor's degree in the same discipline from Sasi Institute of Technology and Engineering. Her technical expertise encompasses a wide range of programming languages, including C, Java, Python, and R, backed by extensive laboratory and teaching experience. She can be contacted at email: syamala@kluniversity.in.