AMERICAN JOURNAL OF MANAGEMENT AND IOT MEDICAL COMPUTING
Peer Reviewed, Referred & Indexed Journal
E-ISSN: 3069-0110
Vol.5, No.2(2026)
www.ajmimc.com

ProbDeepNet: A Cognitively Aligned Severity Intelligence Framework for Latent Bug Semantics in Medical IoT Firmware

E. Mahesh[1], Gurram Raviteja[2], Yerram Harshitha[2], B. Lahari[2], Bolleboina Ranjith[2]

[1]Assistant Professor, [2]UG Student, [1,2]Department of Computer Science and Engineering (Data science)

[1,2]Vaagdevi Engineering College, Bollikunta, Warangal, 506005, Telangana, India

ABSTRACT

Healthcare systems are increasingly vulnerable to cyber-attacks, with over 93 million patient records breached in 2023 and an annual increase of 22% in vulnerability disclosures affecting hospital networks. Manual approaches to bug severity detection are error-prone and inefficient, often failing to manage the growing volume of unstructured medical security reports. To address this, we propose a Natural Language Processing (NLP) framework that leverages a Medical NLP dataset comprising incident reports, advisories, and vulnerability disclosures. The methodology begins with NLP preprocessing and Exploratory Data Analysis (EDA) to clean, normalize, and visualize the dataset. Following this, Lightweight Robustly Optimized bidirectional encoded representation of transformers (LRoBERT) is used with word embeddings for semantic representation of text, ensuring reduced computational cost compared to heavy transformer models. Unlike existing systems based on Stochastic Gradient Descent (SGD) and Natural Gradient Boosting (NGB) Classifier, the proposed pipeline integrates Deep Neural Network (DNN) based feature selection with Natural Gradient Boosting (NGB) model also known as ProbFusionNet to improve predictive performance. The system classifies bug severity into two categories: Normal and Severity, providing actionable insights for prioritizing critical vulnerabilities. By combining contextual embeddings with advanced feature selection, the proposed approach enhances accuracy, reduces misclassification, and ensures faster real-time response. This innovation contributes to strengthening the cybersecurity posture of the healthcare ecosystem by delivering an efficient and scalable solution for automated bug severity detection.

Keywords: Bug Severity Detection, Healthcare Cybersecurity, Natural Language Processing (NLP), LRoBERT, Deep Neural Networks (DNN).

1.INTRODUCTION

With the constant expansion of modern software, its reliability has become questionable, as these programs are prone to many problems and even failure. Software bugs are one of the enduring issues in the software development process. These are created in software during the development process by a programmer's mistake, such as memory flow issues, run-time issues, and deviations from the pre-coordinated running directions [1]. These software bugs can trigger significant security issues, particularly in the area where the fault tolerance rate is low. For example, the Heart Bleed Bug attacked the encryption systems of the software industry in 2015. Therefore, the bug's presence proved quite costly. As the budget of the software industry is limited, it is helpful first to track a bug and evaluate its severity at the right time. In standard practice, the end user experiences a bug while dealing with the system and reports the bug in the bug tracking system (BTS) by entering the severity level, description, product, platform, and component fields. The developer (triager) uses this information to resolve the costly bug, which is a time-consuming process. The triager has to choose and identify the issue from a large number of communicated software bugs [2]. Different bugs have different impacts upon the quality, based on the functionality of software. The triager

(test engineer or developer) assigns the different severity classes to the bug report, as shown in Fig 1 Not all bugs relate to critical issues, as some are frivolous and appeal for an improvement [3]. Let us take an example of an email service provider (ESP). If the system crashes or tosses the error message in spite of signing in with the correct password and user name, such a bug is classified as a critical bug, as it makes the entire application unusable. Further, if there is a major functionality issue, such as when the carbon copy (CC) section of the ESP does not allow adding more than one recipient, then it is also classified as a major bug because it makes the application dysfunctional. The "Terms and Conditions" option in the ESP has multiple links, and if one link is not working properly among the various links, then it is labeled as a minor bug. It does not affect the usability of the application and if there is a spelling mistake in the "license page" of the ESP, this type of issue is classified as a low priority bug.

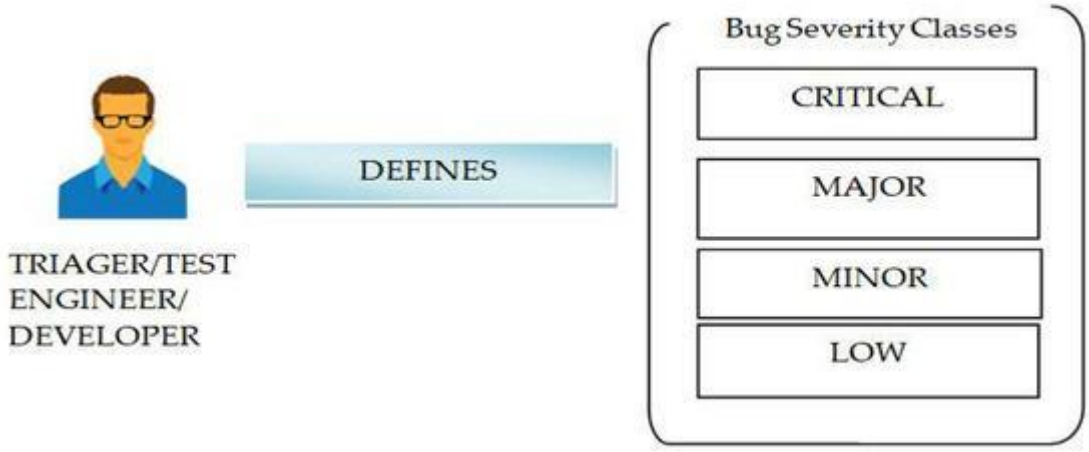


Fig. 1: The basic classes of bug severity.

In the case of numerous bug reports, the manual severity classification is a very dull, laborious, and time-consuming task. Manual bug severity classification is performed by utilizing the bug report content (summary and description). Therefore, bug severity classification techniques that automatically classify the bugs according to comments have generally been utilized by developers to dispense their limited resources.

On the other hand, the bug severity also acts as a dispute between user and developer which requires prompt consideration and resolution. It is a fundamental characteristic that gives general advantages to the industry, which requires earnestness in its resolution with respect to resources. It proves that 80% of bug reports goes through severity reassignment, even after the bug report has been sent to a developer [4]. Additionally, it demonstrates that bug reports with severity reassignment have fundamentally longer resolution times than those with no reassigned severity. These bug severity fields are utilized by the triager to assign the appropriate development groups shown in fig 1. The inaccurate severity assignment essentially postpones the processing time of the bug report, thus influencing the efficiency of developers' work [5]. Numerous bug severity classification techniques have been proposed. Classification models extract the bug features from bug reports, which act as inputs and use various machine learning algorithms for training purposes. After that, the model is used to classify the severity of a new bug report.

2.LITERATURE SURVEY

Kukkar et al. [6] proposed a novel deep learning model for multiclass severity classification called Bug Severity classification to address these challenges by using a Convolutional Neural Network and Random

forest with Boosting (BCR). This model directly learns the latent and highly representative features. Initially, the natural language techniques preprocess the bug report text, and then n-gram is used to extract the features. Further, the Convolutional Neural Network extracts the important feature patterns of respective severity classes. Lastly, the random forest with boosting classifies the multiple bug severity classes. The average accuracy of the proposed model is 96.34% on multiclass severity of five open-source projects. The average F-measures of the proposed BCR and the existing approach were 96.43% and 84.24%, respectively, on binary class severity classification. The results prove that the proposed BCR approach enhances the performance of bug severity classification over the state-of-the-art techniques.

Dao et al. [7] proposed a deep learning framework called MASP that uses convolutional neural networks (CNN) and the content-aspect, sentiment-aspect, quality-aspect, and reporter-aspect features of bug reports to improve prediction performance.

Köksal et al. [8] presented automated bug classification approach applied and validated in an industrial case study. In contrast to earlier studies, our study is applied to a commercial software system based on unstructured bilingual bug reports written in English and Turkish. The presented approach adopts and integrates machine learning (ML), text mining, and natural language processing (NLP) techniques to support the classification of software bugs. The approach has been applied within an industrial case study. Compared to manual classification, our results show that bug classification can be automated and even performs better than manual bug classification.

Albattah et al. [9] investigated eight well-known machine learning and deep learning algorithms for software bug prediction. Their study used a large dataset collected from five publicly available bug datasets that includes about 60 software metrics.

Qian et al. [10] provided a comprehensive investigation and survey of the recent developments in bug deduplication and triage. The study begins by outlining the roadmap of the existing literature, including the research trends, mathematical models, methods, and commonly used datasets in recent years. Subsequently, the paper summarizes the general process of the methods from two perspectives—runtime information-based and bug report-based perspectives—and provides a detailed overview of the methodologies employed in relevant works.

Tabassum et al. [11] proposed a novel hybrid approach based on natural language processing (NLP) and machine learning. To address these issues, the intended outcomes are multi-class supervised classification and bug prioritization using supervised classifiers. After being collected, the dataset was prepared for vectorization, subjected to exploratory data analysis, and pre-processed. The feature extraction and selection methods used for a bag of words are TF-IDF and word2vec. Machine learning models are created after the dataset has undergone a full transformation.

Bhakar et al. [12] presented a comprehensive review of recent approaches for identifying disease severity levels using computational intelligence-based approaches. They followed the PRISMA guidelines and compiled several works related to the severity identification of multidisciplinary diseases of the last decade from well-known publishers, such as MDPI, Springer, IEEE, Elsevier, etc. This article is devoted toward the severity identification of two main diseases, viz. Parkinson's Disease and Diabetic Retinopathy.

Ji, J. et al. [13] investigated the impact of stop word removal on three core bug report classification tasks. The analysis uses a dataset containing over 1.9 million bug reports from eight large-scale open-source projects, including Eclipse, FreeBSD, GCC, Gentoo, Kernel, RedHat, Sourceware, and WebKit. Five deep

learning models are applied: convolutional neural networks, long short-term memory networks, gated recurrent units, Transformers, and BERT. Each model is evaluated on its performance with and without stop word removal during preprocessing.

Liu, J et al. [14] proposed a method that extracts node features and basic path features from the program dependency graph and designs high-severity contextual defect confirmation labels combined with contextual features. Finally, an optimized Support Vector Machine is used to train the automatic detection model for high-severity defects. Their study uses open-source programs to manually implant defects for high-severity defect confirmation verification.

Khandakar, et al. [15] used an available labelled diabetic thermogram dataset and uses the k-mean clustering technique to cluster the severity risk of diabetic foot ulcers using an unsupervised approach. Using the plantar foot temperature, the new clustered dataset is verified by expert medical doctors in terms of risk for the development of foot ulcers. The newly labelled dataset is then investigated in terms of robustness to be classified by any machine learning network. Classical machine learning algorithms with feature engineering and a convolutional neural network (CNN) with image-enhancement techniques are investigated to provide the best-performing network in classifying thermograms based on severity.

3. PROPOSED SYSTEM

The proposed system for bug severity detection in healthcare environments automates the process of analysing medical security data, classifying bug severity, and providing actionable insights. It leverages Natural Language Processing (NLP) to preprocess and extract features from unstructured text data, then applies advanced machine learning models to classify the severity of vulnerabilities. This system aims to improve both the accuracy and efficiency of detecting and prioritizing critical vulnerabilities, ensuring timely and effective responses to security threats as shown in fig 2.

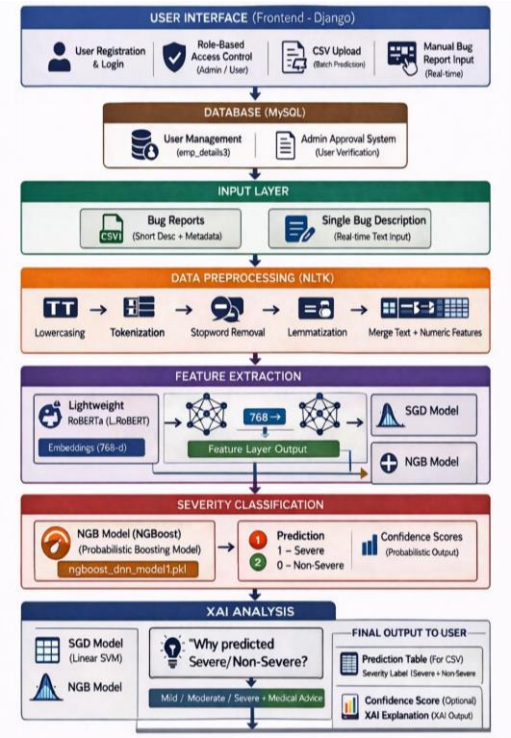


Fig. 2: Proposed system architecture for bug severity detection.

Step 1: Data Collection Medical NLP Dataset: In the first step of the process, the system gathers raw data from various medical sources, including security incident reports, vulnerability disclosures, and advisories. This data is typically unstructured and consists of textual reports generated from different healthcare entities, including hospitals, insurance providers, and other healthcare systems. The raw data needs to be collected, centralized, and organized into a format that can be used for further processing.

- **Data Source Identification:** The data is sourced from multiple channels, such as security advisories, vulnerability reports, and incident logs that describe potential security flaws within healthcare systems.
- **Dataset Preparation:** After collection, this unstructured data is converted into a structured format, like a CSV or database, ensuring it is ready for subsequent processing steps.

The goal of this stage is to prepare the medical NLP dataset for cleaning and preprocessing in the next step, ensuring all relevant information is available for feature extraction.

Step 2: NLP Preprocessing & Exploratory Data Analysis (EDA): Once the dataset is collected, it undergoes an NLP preprocessing phase. This step involves several text-cleaning operations, where the raw textual data is normalized and made ready for analysis. Additionally, Exploratory Data Analysis (EDA) is performed to gain insights into the dataset's structure, uncover patterns, and understand its characteristics. These steps are essential for making the data suitable for feature extraction and model training.

- **Text Cleaning:** The data is processed to remove noise such as irrelevant characters, formatting issues, and stopwords. Text is then tokenized into individual words, and each word is normalized through lemmatization, which reduces it to its root form (e.g., "running" becomes "run").
- **Exploratory Data Analysis (EDA):** The system analyzes the dataset to identify trends, common phrases, and other important features. This involves generating word clouds, histograms of document lengths, and exploring the frequency of terms (such as n-grams and part-of-speech tags). These visualizations provide useful insights that guide the subsequent stages of feature extraction.

This preprocessing step ensures the data is cleaned and transformed into a form that can be used to train machine learning models, providing a solid foundation for feature extraction.

Step 3: Feature Extraction Using Lightweight RoBERT with Word Embeddings

In the feature extraction step, the system uses Lightweight RoBERT (Robustly Optimized BERT) to convert the pre-processed text into meaningful feature vectors, known as word embeddings. RoBERTa is an optimized version of BERT (Bidirectional Encoder Representations from Transformers) that can generate contextualized word representations. These embeddings capture the semantic meaning of the text, which is crucial for the accurate classification of bug severity.

- **RoBERT Model:** The Lightweight RoBERT model is applied to the preprocessed text to generate word embeddings. These embeddings help the model understand the meaning of words within their context, improving the system's ability to differentiate between normal and severe bugs based on the report descriptions.
- **Feature Representation:** The embeddings are processed and aggregated into fixed-length feature vectors, ready to be used in machine learning models. The system may also include non-textual

features (such as metadata or numerical values) if available, ensuring that the model has all the relevant information for classification.

This step transforms raw text data into structured feature vectors, which will then be used to train machine learning models for severity classification.

Step 4: Model Training & Classification: Once the features have been extracted, the system applies machine learning models to classify bug severity into two categories: Normal or Severity. In this phase, both existing and proposed models are tested to evaluate which performs best on the dataset. The models trained include SGD Classifier, NGB, and an enhanced version of NGB that integrates DNN-based feature selection.

- **Existing Models (SGD & NGB):** The system first evaluates the performance of existing models like SGD (Stochastic Gradient Descent) and NGB (Natural Gradient Boosting). These classifiers are trained using the features extracted in Step 3 to determine how well they can predict the severity of bugs in medical datasets.
- **Proposed Model (DNN Feature Selection + NGB):** The proposed model incorporates DNN (Deep Neural Network)-based feature selection, which enhances the feature selection process. The DNN identifies which features are most important for predicting bug severity, helping the NGB model perform better by focusing on the most relevant information.

The goal of this step is to fine-tune the models to ensure they can classify the severity of bugs accurately and efficiently, with the proposed model expected to outperform the traditional models in terms of prediction accuracy.

Step 5: Bug Severity Classification: After the models have been trained, they are used to classify bug severity in new, unseen data. The classification process involves using the trained classifiers (either the NGB or the enhanced DNN + NGB model) to predict whether a bug is of normal severity or requires urgent attention.

- **Prediction:** The models are tested on the validation dataset, where they predict the severity of each vulnerability or bug described in the medical reports.
- **Output:** The predictions are made into binary labels: Normal for low-severity bugs and Severity for high-priority bugs. These labels are used to determine which bugs need immediate attention and which can be handled later.

This classification step provides clear outputs that can be used to prioritize security vulnerabilities within the medical systems, ensuring that the most critical issues are addressed first.

Step 6: Real-Time Response & Actionable Insights: Finally, once the bug severity is classified, the system generates real-time actionable insights to help cybersecurity teams prioritize their responses. These insights are based on the severity classifications from the previous step.

- **Real-Time Analysis:** The system continuously processes new medical security reports, classifying bugs as they arise. This real-time processing ensures that vulnerabilities are identified and addressed as quickly as possible.
- **Actionable Insights:** The system generates alerts or notifications indicating which bugs are of high severity, suggesting that they should be addressed immediately. The insights also include

recommendations for fixing or mitigating the vulnerabilities, enabling cyber security teams to act swiftly.

This final step ensures that healthcare organizations can quickly respond to security threats, reducing the risk of data breaches or other cyber security issues that could compromise patient safety or data privacy.

4. RESULTS ANALYSIS

The results analysis section evaluates the performance and effectiveness of the proposed system in achieving accurate and reliable outcomes. It focuses on assessing the model using various evaluation metrics such as accuracy, precision, recall, and F1-score to ensure comprehensive performance measurement. The analysis also compares the proposed approach with existing methods to highlight improvements and advantages. Graphical representations and visualizations are utilized to clearly interpret the results and identify patterns or trends. Additionally, the robustness and generalization capability of the model are examined using test datasets. This section provides critical insights into the strengths and limitations of the system, ensuring its suitability for real-world applications.

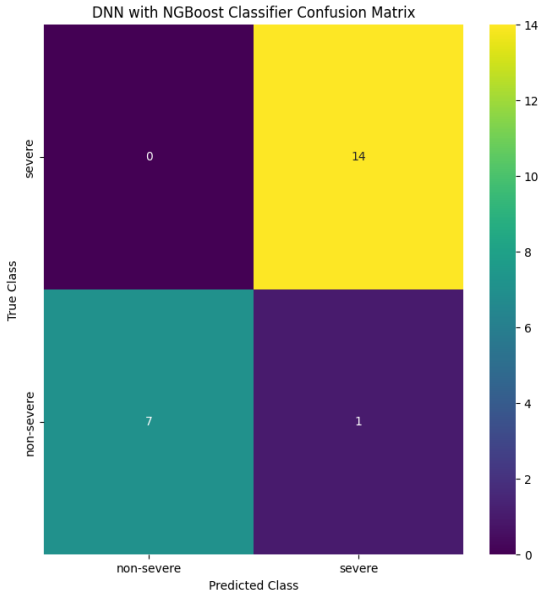


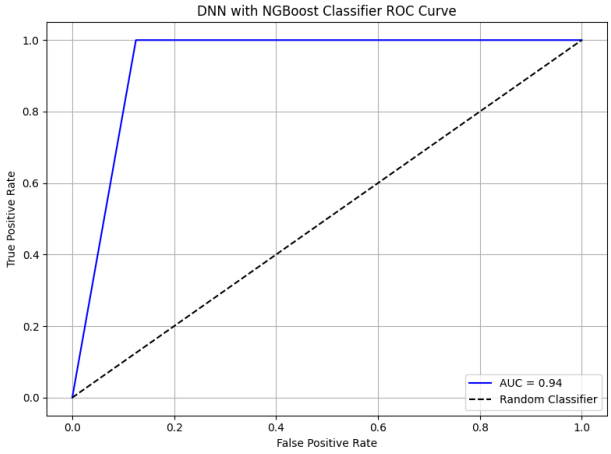
Fig. 3: Confusion matrix obtained using ProbFusionNet Classifier.

Fig 3 illustrates the confusion matrix of the ProbFusionNet classifier, representing the classification performance between severe and non-severe classes. It depicts that all severe instances are misclassified as non-severe, with zero true positives and a high number of false negatives, indicating poor detection of severe cases. Conversely, the model correctly identifies a limited number of non-severe instances, while also misclassifying several as severe, reflecting imbalance in prediction capability. The distribution highlights significant class-wise performance disparity, suggesting that the model is biased toward the non-severe class and struggles to generalize effectively for severe condition identification.

Fig 4 illustrates the ROC curve of the ProbFusionNet classifier, demonstrating the trade-off between true positive rate and false positive rate across different threshold settings. It depicts a steep rise in sensitivity, indicating that the model achieves high true positive rates with minimal false positives. The curve remains significantly above the diagonal baseline, reflecting strong discriminative capability compared to random

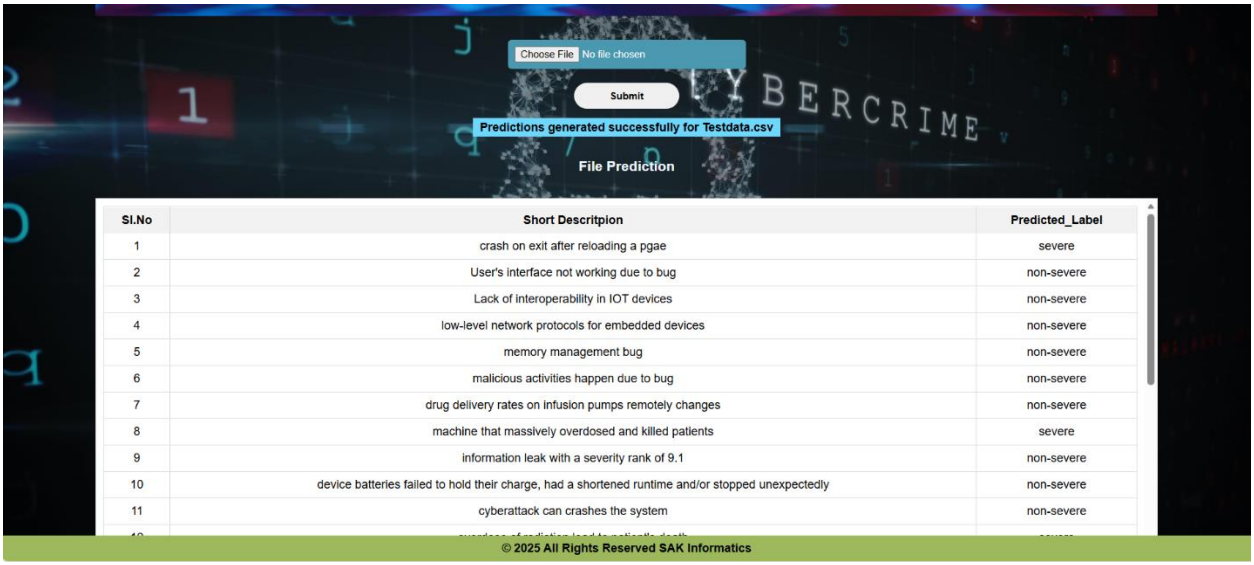
AMERICAN JOURNAL OF MANAGEMENT AND IOT MEDICAL COMPUTING
Peer Reviewed, Referred & Indexed Journal
E-ISSN: 3069-0110
Vol.5, No.2(2026)
www.ajmimc.com

classification. The reported AUC value of 0.94 highlights the model's high overall classification performance and its effectiveness in distinguishing between severe and non-severe classes.



(d)

Fig. 4: ROC Curve obtained using ProbFusionNet Classifier.



Sl.No	Short Descritpion	Predicted_Label
1	crash on exit after reloading a pgae	severe
2	User's interface not working due to bug	non-severe
3	Lack of interoperability in IOT devices	non-severe
4	low-level network protocols for embedded devices	non-severe
5	memory management bug	non-severe
6	malicious activities happen due to bug	non-severe
7	drug delivery rates on infusion pumps remotely changes	non-severe
8	machine that massively overdosed and killed patients	severe
9	information leak with a severity rank of 9.1	non-severe
10	device batteries failed to hold their charge, had a shortened runtime and/or stopped unexpectedly	non-severe
11	cyberattack can crashes the system	non-severe

Fig. 5: Predictions generated for the test data.

Table 1: Performance comparison of all Classifier models.

Algorithm	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
SGD	45.455	36.458	38.393	37.143
NGB	63.636	58.333	55.357	54.167
DNN with SGD	68.182	83.333	56.250	51.111
ProbFusionNet	95.455	96.667	93.750	94.943

Fig 5 shows the file upload interface through which a test dataset is submitted for analysis, followed by a confirmation message indicating successful prediction generation. Below this, a tabular output is presented containing the Serial Number, Short Description of the bug, and the corresponding Predicted Label. Each medical or technical bug description is automatically classified as either severe or non-severe based on the learned contextual patterns from the trained Lightweight ProbFusionNet model. This visualization demonstrates the system's ability to process real-world medical cybersecurity data and generate clear, actionable severity predictions that can be used to prioritize critical vulnerabilities efficiently.

Table 1 presents a comprehensive performance comparison of four classification models evaluated on the medical bug severity prediction task using key metrics: Accuracy, Precision, Recall, and F1-Score (reported in percentages). The ProbFusionNet Classifier emerges as the top-performing model, achieving an outstanding 95.455% accuracy, 96.667% precision, 93.750% recall, and 94.943% F1-score, demonstrating exceptional balance and reliability in identifying severe cases. In contrast, the SGD Classifier records the lowest performance across all metrics, with only 45.455% accuracy and an F1-score of 37.143%, indicating limited discriminative power. The NGB Classifier and DNN with SGD Classifier show moderate improvements, with F1-scores of 54.167% and 51.111%, respectively, but remain significantly outperformed by the deep ensemble approach combining DNN-extracted features and probabilistic boosting.

5. Conclusion

The proposed system presents a robust and intelligent framework for automated bug severity grading in IoT medical firmware by integrating advanced NLP, deep learning, and probabilistic modeling techniques into a unified pipeline. The combination of LRoBERT for contextual semantic feature extraction and DNN for deep feature refinement enables the system to capture both linguistic patterns and complex representations from bug reports with high precision. The final classification using NGB enhances prediction reliability by incorporating uncertainty-aware decision making, making the system more suitable for critical medical environments where accuracy and confidence are essential. Additionally, the inclusion of baseline models such as SGD and NGB provides a comparative foundation, validating the effectiveness and superiority of the hybrid approach. The system further strengthens its practical applicability through real-time and batch prediction capabilities, seamless integration with a web-based interface, and structured output generation including severity labels and confidence scores. The XAI analysis module ensures interpretability by providing concise human-understandable explanations, thereby improving trust and transparency in automated decisions. The proposed architecture demonstrates a scalable, efficient, and explainable solution that significantly enhances the process of identifying and prioritizing critical bugs in IoT medical firmware systems, contributing to improved software reliability, patient safety, and intelligent decision support in healthcare-driven embedded environments.

REFERENCES

[1] Sharma, Y.; Dagur, A.; Chaturvedi, R. Automated bug reporting system with keyword-driven framework. In Soft Computing and Signal Processing; Springer: Singapore, 2019; pp. 271–277.

[2] Canfora, G.; Cerulo, L. How software repositories can help in resolving a new change request. In Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, Budapest, Hungary, 24–25 September 2005; pp. 89–99.

[3] Antoniol, G.; Ayari, K.; Di Penta, M.; Khomh, F.; Guéhéneuc, Y.G. Is it a bug or an enhancement?: a text-based approach to classify change requests. In Proceedings of the ACM 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, Ontario, Canada, 27–30 October 2008; p. 23.

[4] Angel, T.S.; Kumar, G.S.; Sehgal, V.M.; Nayak, G. Effective bug processing and tracking system. J. Comput. Theor. Nanosci. 2018, 15, 2604–2606.

[5] Nagwani, N.K.; Verma, S.; Mehta, K.K. Generating taxonomic terms for software bug classification by utilizing topic models based on Latent Dirichlet Allocation. In Proceedings of the IEEE 11th International Conference on ICT and Knowledge Engineering (ICT&KE), Bangkok, Thailand, 20–22 November 2013; pp. 1–5.

[6] Kukkar, A.; Mohana, R.; Nayyar, A.; Kim, J.; Kang, B.-G.; Chilamkurti, N. A Novel Deep-Learning-Based Bug Severity Classification Technique Using Convolutional Neural Networks and Random Forest with Boosting. Sensors 2019, 19, 2964. https://doi.org/10.3390/s19132964

[7] Dao, A.-H.; Yang, C.-Z. Severity Prediction for Bug Reports Using Multi-Aspect Features: A Deep Learning Approach. Mathematics 2021, 9, 1644. https://doi.org/10.3390/math9141644

[8] Köksal, Ö.; Tekinerdogan, B. Automated Classification of Unstructured Bilingual Software Bug Reports: An Industrial Case Study Research. Appl. Sci. 2022, 12, 338. https://doi.org/10.3390/app12010338

[9] Albattah, W.; Alzahrani, M. Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach. AI 2024, 5, 1743-1758. https://doi.org/10.3390/ai5040086

[10] Saai Reddy Purmani, S. (2023). The Transformation of IT Leadership in Business Organizations: Shifting from Technical Supervision to Strategic Empowerment. JOURNAL OF ADVANCE AND FUTURE RESEARCH, 1(5). https://doi.org/10.56975/jaafr.v1i5.503885

[11] Tabassum, N.; Namoun, A.; Alyas, T.; Tufail, A.; Taqi, M.; Kim, K.-H. Classification of Bugs in Cloud Computing Applications Using Machine Learning Techniques. Appl. Sci. 2023, 13, 2880. https://doi.org/10.3390/app13052880

[12] Bhakar, S.; Sinwar, D.; Pradhan, N.; Dhaka, V.S.; Cherrez-Ojeda, I.; Parveen, A.; Hassan, M.U. Computational Intelligence-Based Disease Severity Identification: A Review of Multidisciplinary Domains. Diagnostics 2023, 13, 1212. https://doi.org/10.3390/diagnostics13071212

[13] Ji, J.; Yang, G. Do Stop Words Matter in Bug Report Analysis? Empirical Findings Using Deep Learning Models Across Duplicate, Severity, and Priority Classification. Appl. Sci. 2025, 15, 9178. https://doi.org/10.3390/app15169178

[14] Liu, J.; Liang, C.; Feng, J.; Xiao, A.; Zeng, H.; Wu, Q.; Yu, T. A Multi-Feature Fusion-Based Automatic Detection Method for High-Severity Defects. Electronics 2023, 12, 3075. https://doi.org/10.3390/electronics12143075

[15] Khandakar, A.; Chowdhury, M.E.H.; Reaz, M.B.I.; Ali, S.H.M.; Kiranyaz, S.; Rahman, T.; Chowdhury, M.H.; Ayari, M.A.; Alfkey, R.; Bakar, A.A.A.; et al. A Novel Machine Learning

Approach for Severity Classification of Diabetic Foot Complications Using Thermogram Images. Sensors 2022, 22, 4249. https://doi.org/10.3390/s22114249