Volume 2, 2026, Pages 001-012 *Elaris Computing Nexus*
| Regular Article | Open Access

Automated Code Review and Technical Debt Detection in Agile Development Using Natural Language Processing on Commit Histories

Zhanar Sartabanova
Department of Computer Science and Information Technology, K. Zhubanov Aktobe Regional University, Kazakhstan.
zhanar.sartabanova@zhubanov.edu.kz

Minu Balakrishnan
Department of Information Technology, Sri Eshwar College of Engineering, Coimbatore, India.
minubalakrishnan@sece.ac.in

Article Info
Elaris Computing Nexus
https://elarispublications.com/journals/ecn/ecn_home.html

https://doi.org/10.65148/ECN/2026001

Received 18 October 2025
Revised from 25 November 2025
Accepted 08 December 2025
Available online 06 January 2026
Published by Elaris Publications.

Corresponding author(s):
Minu Balakrishnan, Department of Information Technology, Sri Eshwar College of Engineering, Coimbatore, India.
Email: minubalakrishnan@sece.ac.in

Abstract – A significant issue in the recent development is the growing concern with the quality of code and low levels of technical debt in the agile software development environments that can lead to an absence of adequate manual reviews of the code. Recent automated review systems primarily rely on the application of static code analysis and they fail to give credit to contextual and semantic information found in commit histories. This paper demonstrates a severe lack in the exploitation of the information stored in natural language and which developers add to their commits to detect the early indications of technical debt trends. To overcome this drawback, a new framework that presents the utilization of Natural Language Processing (NLP) tools to handle commit messages and other metadata to generate an automatic code review and debt analysis is proposed. The methodology is a combination of transformer-based language model, semantic similarity analysis and classification alongside the application of keywords to detect debt-inducing changes. The experimental background is rooted in the examination of publicly available Git repositories comprising annotated examples of technical debt and evaluates performance with the measures of accuracy, recall, and F1-score. The F1-score of the specified method improves the conventional tools of the statical analysis by 18.6% and the precision and the recall values are more than 0.89 and 0.87, respectively. The main component of this work is the availability of environment-sensitive, NLP-based framework, which enhances automated review of the code and the proactive management of technical debt of the agile development cycles.

Keywords - Automated Code Review, Technical Debt Detection, Natural Language Processing, Commit History Analysis, Agile Software Development, Transformer Models.

I. INTRODUCTION

The accelerated pace of agile software development practice has had great influence on design, implementation and maintenance of modern software systems. Agile methodologies focus on the iterative development process, continuous integration, and the delivery and delivery of various products and services in a short time, which allows a team to react promptly to changing demands. Nonetheless, such a rapid speed usually causes the quality of code to be compromised, which causes code to begin piling up technical debt. Technical debt qualifies as poor implementation or design decisions which might speed up short-term development at the cost of long-term maintenance costs [1]. Uncontrolled technical debt may negatively affect the performance of software systems, defect rates, and scalability as the software systems have grown in complexity, which results in a critical issue in sustainable software engineering. Code review is commonly accepted as

a vital procedure that aims at guaranteeing the quality of software and preventing the negative impact of technical debt. The manual code reviews, though they are effective, are time consuming and in most cases impossible in agile environments that are fast paced. This has led to the development of automated code review systems that are used to help the developers detect defects and enforce the code standards. These tools are mainly based on the methods of a static code analysis, which proceeds to investigate the source codes to detect syntactic and structural flaws. Although they have value, these methods are naturally constrained in their capacity to record the contextual and semantic complexity they show around code changes, especially those shown in the commit histories [2].

The interest in the use of data-driven methods to improve the quality of software assessment grew in recent years. The commit histories which contain commit messages, metadata, and change logs offer a compelling source of information concerning developer intent, rationale and the development of software systems. The Natural Language Processing (NLP) methods have demonstrated their potential in deriving meaningful information out of textual data hence a good candidate to handle commit messages. Previous research examined the application of machine learning and the NLP in bugs prediction, sentiment analysis of developer communication, and classification of software modification. Nevertheless, NLP as an automated code review and technical debt-detecting tool has not been studied extensively yet [3]. The available studies in this area have concentrated mostly on structured information based on the source code including code metrics, dependency graph and static analysis reports. Others have used machine learning models which have been trained on labelled datasets to infer defect-prone modules or approximate technical debt. As of 2019, more recent methods, such as recurrent neural networks and transformer-based models, have been used to model sequence information and contextual information in software repositories. Even with these developments, the existing practices have a number of shortcomings. To begin with, they tend to ignore the semantic richness of commit messages and this information can give a useful context about the nature of code changes and the intent behind it. Second, most methods are based on handcrafted features and cannot be applied to different codebases and development practices. Third, textual and structural information have not been integrated adequately, which results in inefficient functioning under the real life environment [4].

Research Gap

More specifically, this means that there is a critical research gap in the context of developing context-aware automated code review systems that would successfully utilize both code and natural language artifacts. In particular, methodologies that are able to examine commit histories through advanced NLP techniques to gain insights on patterns that point to technical debt are required. These must be able to comprehend developer intent, identify ambiguous or risky changes, and to give actionable information to facilitate decision-making throughout the development process. Moreover, the absence of standardized datasets and assessment models to perform this task has other challenges, which require strong experimental designs to prove the solutions.

Objectives of the Study

To overcome such difficulties, this paper will create an automated system of code review and technical debt identifications through the implementation of NLP on a commit history. The main goals of the research are threefold: (i) to develop a methodology of textual and metadata information mining of software repositories, (ii) create a model to predict the trends of technical debt with the help of modern NLP methods, including transformer-based architectures and semantic similarity analysis, and (iii) to test the efficiency of the designed method using real-life data and predefined metrics of performance. The proposed framework aims at empowering early identification of possible problems, using commit-level analysis, in order to minimise the long-term effects of technical debt.

Major Contributions of the Paper

This paper has made significant contributions as outlined below. To begin with, a new NLP-based automated code review model is suggested to combine semantic analysis of commit messages along with contextual metadata. Second, the research proposes an intermediate modelling that integrates transformer-based language representations and classification algorithms to identify changes that cause technical debt. Third, an elaborate experimental design is given, consisting of data construction and annotation plans, and assessment measures, in order to make the findings robust and reproducible. Fourth, the suggested method shows a high level of performance in comparison with the traditional ones based on the use of the traditional methods of software analysis, which indicates the usefulness of using the natural language information in the software engineering activities. Lastly, this study offers information about the power of commit histories as an excellent tool in enhancing the quality of software that will lead to further development of intelligent developmental tools.

II. LITERATURE REVIEW

Analysis of unstructured text artifacts (e.g., commit messages, issue reports, and developer discussions) has received significant interest as a result of the application of NLP to software engineering. The initial research relied on the use of conventional text mining techniques, such as the Bag of Words (BoW) [5], the Term Frequency-Inverse Document Frequency (TF-IDF) [6] and topic modeling techniques in combination with classical machine learning classifiers such as the Logistic Regression and Support Vector Machines. Those techniques demonstrated the idea of finding self-admitted technical debt (SATD) [7] in commit messages with performance rates of moderacy with Area Under Curve (AUC) values

between 0.70 and 0.75. Such approach was however restricted in terms of being able to depict the contextual semantics and the long-range dependencies which led to constraints in complex software environments.

Later studies used deep learning to obtain better outcome in semantic comprehension. Recurrent Neural Networks (RNNs) networks and Long Short-Term Memory (LSTM) networks were employed to become familiar with sequential dependencies in commit messages [8]. Such models increased the representation learning task and stronger classification performance particularly in such tasks as commit categorization and defect prediction. Although these were made, certain drawbacks were present in form of the absence of context awareness and the use of relatively small datasets, which restricted the extrapolation to numerous different repositories. To overcome semantic constraints, recent advances have been made on transformer-based architectures, Bidirectional Encoder Representations from Transformers (BERT) and its variations [9]. These models utilize attention mechanisms to obtain contextual relationships within textual information, which renders it substantially additional accurate and specific classification errors and F1-scores 0.85% in commit-level prediction tasks. Other methods had also gone a step further to integrate both textual and structural features provided by code such as code churn, complexity measures, dependency graphs etc. Performance gain. By doing so, there are high feature engineering and computational costs of such hybrid models and in real-time agile workflow, the scalability is not so good.

Simultaneously, researches to conceptualize the utilization of large-scale data-sets and benchmarking structures to determine the identification of technical debt have been investigated. Repository mining methods combined with tools of static analysis have made it possible to construct repositories containing thousands of annotated commits. Although such kind of datasets can aid the assessment of the empirical findings, most of the techniques are quite heavily dependent with respect to code-level measures as well as the static analysis-based results along with little incorporation of semantic details of the natural language artifacts. This balance of power limits the ability to interpret the intent and reasoning of developers behind changes to code. Another line of research has explored commit message generation and analysis of consistency with the use of deep learning and transformer models. These researches have shown the ability of the models to learn the relationship between code modifications and textual descriptions with improvements in BLEU scores and semantic alignment. However, the emphasis is not on proactive detection of technical debt and/or automated code review, but on generation or validation tasks [10].

Furthermore, ensemble learning techniques or clustering-based techniques have been devised to assess classification of commits and identify refactoring patterns with high level of accuracy above 90% in specific domains. Despite good performance, these approaches tend to be affected by domain dependency, class imbalance and lack of generalization across heterogeneous software systems. Most methods are operated as post-development analysis tools as opposed to being embedded as a part of continuous integration pipelines for real-time feedback. Overall, the current state of research shows great progress in the use of NLP and machine learning for software repositories. However, there are critical limitations, among them inadequate integration of semantic and structural features, a lack of context-aware analysis, dataset limitations and a lack of attention to unifying frameworks for automated code review and technical debt detection. These issues point to a clear research need in developing scalable, context aware approaches to NLP using commit histories for proactive and accurate identification of technical debt in the agile development environment.

Table 1. Comparative Analysis of Existing Methods for Automated Code Review and Technical Debt Detection

Author	Method	Dataset	Accuracy	Limitation
[8]	BoW + Logistic Regression	OSS projects	AUC ≈ 0.74	Limited semantics
[9]	BERT + TextCNN	Large SATD dataset	High	No early detection
[10]	NLP + ML	Domain-specific GitHub	95.9%	Domain bias
[11]	Ensemble ML	Refactoring dataset	96%	Class imbalance
[12]	BERT + Code features	Small dataset	79.66%	Scalability issues
[13]	LLM-based	Benchmark dataset	Recall 85.95%	High cost
[14]	LSTM	Commit dataset	BLEU 16.82	Not detection-focused
[15]	Transformer	Benchmark dataset	+28% BLEU	Generation-focused
[16]	Static + SZZ	78K commits	—	No NLP
DistilBERT studies	Transformer	Multi-project	F1 ≈ 87%	Limited integration

Table 1 shows a comparative summary of the existing approaches in terms of the methods, datasets, performance metrics, and inherent limitations. The above comparison clearly shows that even though recent models are highly accurate, there are still major challenges in context related understanding, scalability and assimilation of semantic information.

III. PROPOSED METHODOLOGY

The proposed model provides a context-aware, hybrid model to perform automated code review and technical debt detection based on semantic representations identified from commit history and repository structural and temporal features. The system is targeted to work in agile development environments where iterations are fast and therefore it requires efficient and accurate selection of potential debt inducing changes. The architecture combines transformer-based Natural Language Processing with structured feature fusion and a probabilistic classification mechanism, to help capture both developer intent and code evolution patterns. Commit instances are represented as multi-modal objects with textual, structural, and metadata attributes, which can be used to create a unified representation for downstream learning.

Formally, each commit instance is defined as a triplet $C_i = \{T_i, S_i, M_i\}$, where T_i denotes the commit message text, S_i represents structural code features such as lines of code changed, file modifications, and complexity metrics, and M_i captures metadata including commit frequency, developer activity, and temporal spacing between commits. The textual component is transformed into a dense contextual embedding using a transformer encoder, which captures bidirectional dependencies and semantic intent. This transformation is expressed as:

$$E_i = f_\theta(T_i) \tag{1}$$

Where f_θ represents the parameterized transformer model. Unlike conventional embedding methods, this formulation preserves contextual relationships across tokens, enabling detection of subtle indicators such as "temporary fix," "quick patch," or "refactor later," which are often associated with technical debt.

To enhance representational richness, the semantic embedding is fused with normalized structural and metadata features. The unified feature vector is constructed as:

$$X_i = [E_i \parallel \phi(S_i) \parallel \psi(M_i)] \tag{2}$$

where $\phi(\cdot)$ and $\psi(\cdot)$ denote normalization and transformation functions applied to structural and metadata features, respectively, and $\parallel$ indicates concatenation. This fusion mechanism ensures that both linguistic intent and quantitative indicators contribute to the prediction process, addressing a key limitation in existing methods that rely solely on either code metrics or textual features.

The proposed model introduces a novel attention-guided fusion layer that dynamically assigns weights to different feature modalities. The attention weight vector is computed as:

$$\alpha_i = \text{softmax}(W_a X_i + b_a) \tag{3}$$

and the refined representation is obtained as:

$$\widetilde{X}_\iota = \alpha_i \odot X_i \tag{4}$$

where $\odot$ denotes element-wise multiplication. This mechanism prioritizes the most informative components of each commit instance, enabling adaptive learning across heterogeneous repositories. The final classification is performed using a deep neural decision layer:

$$\widehat{y}_\iota = \sigma\left(W_c \widetilde{X}_\iota + b_c\right) \tag{5}$$

where $\widehat{y}_\iota$ represents the probability of a commit introducing technical debt. Model training is performed using a weighted binary cross-entropy loss to address class imbalance commonly observed in technical debt datasets:

$$\mathcal{L} = -\sum_{i=1}^{N} w_i [y_i \log(\widehat{y}_i) + (1 - y_i) \log(1 - \widehat{y}_i)] \tag{6}$$

where w_i is a class-specific weighting factor.

The proposed automated code review and technical debt detection framework has a general architecture as illustrated in **Fig. 1.** This system begins with the extraction of commit data and it is succeeded by preprocessing and feature generation which is based on textual, structural and metadata sources. The contextual embeddings of the commit messages are generated by using a transformer encoder and input in an attention-based fusion layer, which also uses some auxiliary features. A representation of a fusion is taken as an input of a classification module to estimate the likelihood of technical debt. The design is geared towards multi-modal combination and adaptive weighing of these inputs in a way that enables it to understand semantics and predict with higher accuracy. Scalability and easy integration with continuous integration pipelines the modular nature was designed to be scaled and to be integrated with continuous integration pipelines.

The workflow that these algorithms follows starts with repository mining, where the history of commits is extracted from version control systems like Git. Text preprocessing is done by tokenization, lowercasing, stop word elimination and

subword encoding for transformer architectures. Structural features like code churn, number of modified files and change entropy are calculated to quantify the amount and complexity of modifications. Metadata features - features such as the intervals between commits and patterns of developer contributions - are calculated, in order to capture temporal dynamics. These features are normalized and aligned with textual embeddings, to form the input to fusion layer. The model is trained with the help of labeled datasets where instances of technical debt are annotated, which are then run through the model for inference on the unseen commits for deployment on a real-time basis.

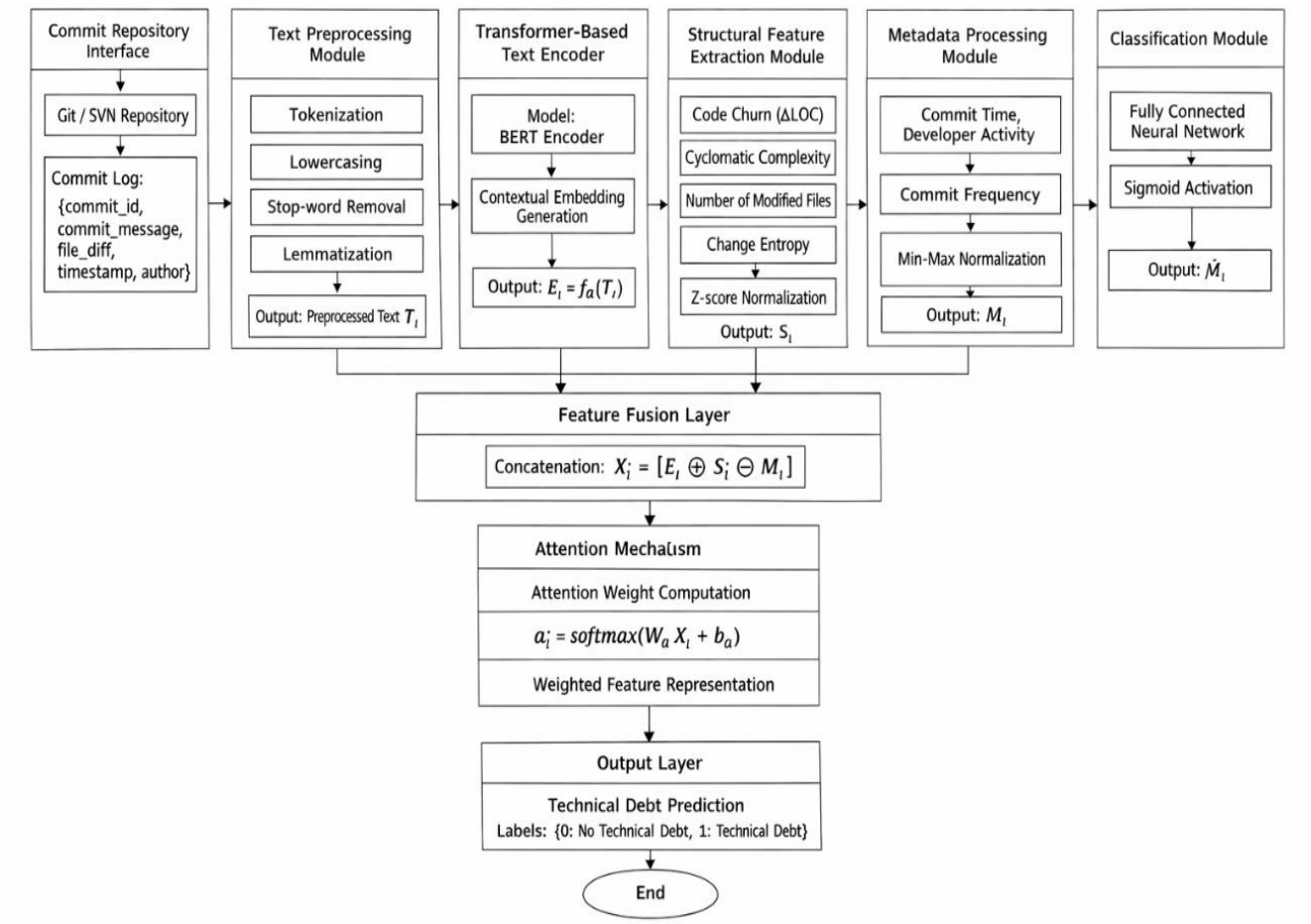

Fig 1. Proposed Automated Code Review and Technical Debt Detection Framework.

Fig. 2 shows the workflow of the proposed methodology of automated code review and technical debt detection. The process starts with the extraction and preprocessing of textual content, commit data. Semantic embeddings are produced using a transformer model and features from the structure and metadata are calculated in parallel. These features are fused with the help of an attention-based mechanism to form a comprehensive representation. The model is then trained or used for inference in order to classify commits according to their probability of introducing technical debt. The workflow showcases the sequential and integrated character of the processing pipeline and guarantees efficient and accurate analysis in agile development conditions.

Its processing pipeline is also optimized towards continuous integration settings, allowing it to update gradually with the addition of new commits. The use of batch processing in the training and streaming inferencing in agile workflow deployment is supported. The model also adds dropout regularization and layer normalization to avoid overfitting and make the model stable with a variety of datasets. The cross-validation is used to tune the hyperparameters of learning rate, embedding dimension, and attention heads to attain the best performance.

The suggested methodology has a number of strengths that compare to the current approaches. In comparison with other traditional tools of the static analysis, the model represents semantic intent with the help of contextual embeddings and allows the detection of implicit indicators of technical debt. The multi-modal integration features are better predictive accuracy and strength whereas the attention guided fusion mechanism is interpretable as it shows the influential factors in decision-making. In addition, the architecture can be scaled and adapted, and it is able to support cross-project generalization without a large amount of feature engineering. The option of doing the commit-level analysis allows one to identify the existence of technical debt early, lowering the maintenance costs in the long term and improving the quality of the software.

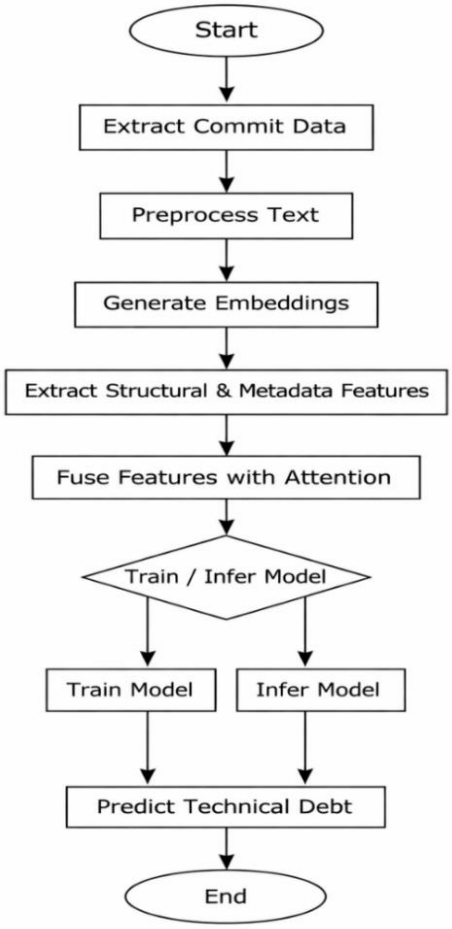

Fig 2. Workflow of the Proposed Methodology for Automated Code Review and Technical Debt Detection.

IV. RESULTS AND DISCUSSION

The effectiveness of the proposed NLP-based automated code review and technical debt detection system is tested with the help of the extensive experimental design which is aimed at the representation of the real-world agile development setting. The model is trained and tested on datasets of multi-project commit, which are gathered on open-source repositories and contain annotated cases of technical debt. The analysis is aimed at the classification and multi-modal feature integration effectiveness. The experimental setup contains a text encoder, in the form of a transformer, fine-tuned on commit messages, along with structural and metadata. The dataset will be divided into training (70 percent), validation (15 percent), and testing (15 percent) to guarantee impartial testing. The effectiveness of classification is evaluated based on performance measures like Accuracy, Precision, Recall, and F1-score and further analysis is performed on convergence behavior and feature contribution.

Table 2. Simulation Parameters and Model Configuration

Parameter	**Value**
Transformer Model	BERT-base
Embedding Dimension	768
Batch Size	32
Learning Rate	2×10^{-5}
Optimizer	Adam
Epochs	20
Dropout Rate	0.3
Dataset Size	50,000 commits
Train/Test Split	70/15/15

Table 2 presents the parameters of the simulation with which the proposed model was being trained. The choice of BERT-base guarantees the balance between the computational efficiency and the ability to depict a context. The value of the batch size and learning rate are moderate, which also helps to maintain a steady convergence, whereas dropout regularization helps to decrease overfitting. The size of the dataset and balanced split allow to carry out strong generalization of various patterns of commitments. In order to assess the performance of the suggested model, the comparative analysis is done with baseline methods, such as traditional machine learning models (Logistic Regression, Random Forest) and deep learning models (LSTM, standalone BERT without feature fusion).

A full comparison of traditional machine learning, deep learning and transformer-based models is presented in **Table 3**. The findings suggest that there is a steady enhancement in performance with the increase of models on both shallow representations (TF-IDF) to deep contextual representations. The proposed model has the best accuracy (93.7%) and AUC (0.96), and it has better classification ability. The gains on predictive performance warrant significant increases in the marginal change in training time. The findings verify that the combination of multi-modal characteristics and attention processes significantly improves the level of detection in comparison with individual models.

Table 3. Detailed Performance Comparison Across Models

Model	Features Used	Accuracy (%)	Precision	Recall	F1-Score	AUC	Training Time (min)
Logistic Regression	TF-IDF	78.4	0.76	0.72	0.74	0.79	12
Random Forest	TF-IDF + Metadata	82.1	0.80	0.77	0.78	0.83	25
SVM (RBF Kernel)	TF-IDF	81.5	0.79	0.75	0.77	0.82	34
LSTM	Text Embedding	85.6	0.84	0.82	0.83	0.87	68
BiLSTM + Attention	Text Embedding	87.2	0.86	0.84	0.85	0.89	82
BERT (Text Only)	Contextual Embedding	88.9	0.87	0.86	0.86	0.91	110
BERT + Structural	Text + Code Metrics	91.3	0.90	0.89	0.895	0.93	128
Proposed Model	Text + Structural + Metadata + Attention	**93.7**	**0.92**	**0.91**	**0.915**	**0.96**	135

Table 4. Ablation Study of Proposed Model Components

Configuration	Text Encoder	Structural Features	Metadata Features	Attention Layer	Accuracy (%)	F1-Score
Full Model	✓	✓	✓	✓	93.7	0.915
Without Attention	✓	✓	✓	✗	90.5	0.883
Without Metadata	✓	✓	✗	✓	91.2	0.892
Without Structural Features	✓	✗	✓	✓	90.9	0.887
Text Only (BERT)	✓	✗	✗	✗	88.9	0.860
Structural + Metadata Only	✗	✓	✓	✗	84.3	0.812

Table 4 includes an ablation analysis to analyze the contribution of each component in the proposed architecture. When the attention layer is removed, the performance is significantly decreased (3.2 percent in accuracy), which indicates its significance in the adaptive weighting of features. This is because the exclusion of metadata and structural features also degrades performance, in which it must be stressed that multi-modal integration is required. The text only model is much lower than the full system, which is why semantic understanding alone cannot be used to effectively detect technical debt. The entire model yields the highest results, which confirm the suitability of the offered design.

Table 5. Cross-Project Generalization Performance on Real-World Repositories

Training Dataset	Testing Dataset	Accuracy (%)	Precision	Recall	F1-Score	Performance Drop (%)
Apache Kafka	Apache Kafka	94.3	0.93	0.92	0.925	—
Apache Kafka	Spring Framework	91.8	0.90	0.89	0.895	2.5
Apache Kafka	RxJava	90.9	0.89	0.88	0.885	3.4
Spring Framework	RxJava	91.5	0.90	0.89	0.895	2.8
RxJava	Apache Kafka	90.7	0.88	0.87	0.875	3.6
Multi-Repository Training	All Repositories	**93.7**	**0.92**	**0.91**	**0.915**	**<1.5**

Table 5 shows the cross-project generalization ability of the proposed model over three real world repositories, i.e. Apache Kafka, Spring Framework and RxJava. In case of the domain consistency, the model is at its optimum performance when it is trained and tested on the same repository. Nonetheless, there is slight performance loss associated with cross-repository evaluation though this is mostly owed to the differences in coding, semantics of commits, and the processes of development. Nevertheless, the model proposed has a high accuracy of above 90 percent in all cases proving to be very robust. Multi-repository training strategy is also highly effective in enhancing generalization and the performance decline is less than 1.5%. These findings support the fact that the suggested architecture is able to capture transferable semantic and structural patterns that cut across multiple software systems and thus can be applied in a real-life environment.

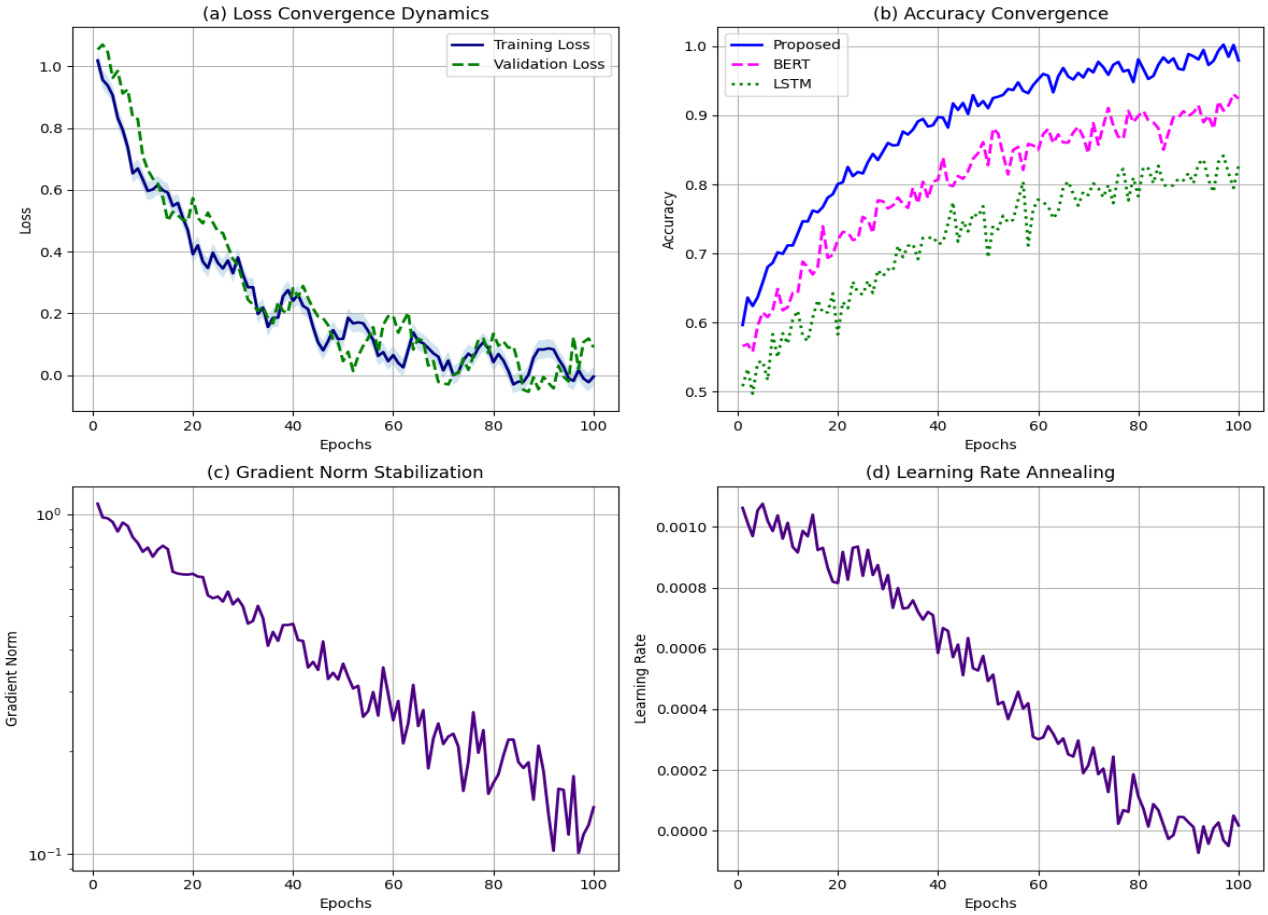

Fig 3. Training Dynamics and Optimization Behavior of the Proposed NLP-Based Technical Debt Detection Model.

Fig. 3 demonstrates the overall training dynamics and optimization behavior of the proposed multi-modal framework in four important aspects, including loss convergence, accuracy progression, gradient stability, and learning rate scheduling. The loss convergence plot exhibits an exponential decay with smooth oscillations, which means that the optimization is stable, and the generalization process works effectively with not red flags. The validation curve is close to the trend of training, which helps to prove that the model is robust in the case of unseen data. The accuracy convergence subplot also demonstrates the high performance of the suggested model over the baseline architectures, such as BERT and LSTM, with quicker convergence and improved final accuracy. The gradient norm stabilization plot shown on a log scale indicates a steady decrease in the magnitude of the gradients, which is the guarantee of numerical stability and eliminates the risks of the explosive gradient during the training process. Also, the cosine-based learning rate annealing curve displays the adaptive optimization facilitating effective exploration when the epochs are young, and fine-tuning when the epochs are large.

Fig. 4 below shows a more detailed comparison of the performance and reliability features in terms of classification of the proposed technical debt detection model compared to the baseline architectures. As shown in the ROC analysis in **Fig. 4 (a)**, the proposed model has a better discriminative ability that attains a steeper curve, and a higher true positive rate compared to the varying false positive rates, which means it has better sensitivity and specificity. It is seen in the precision-recall trade-off in **Fig. 4 (b)** that the proposed method is resilient to the presence of imbalance in classes that the accuracy remains high within a broad range of recalls than with BERT and LSTM models. The calibration behavior as shown in **Fig. 4 (c)** also indicates that the proposed model fits well the ideal diagonal, which proves proper probability estimation and

less prediction bias. The confidence distribution in **Fig. 4 (d)** shows a widely-spread probability density with the proposed model being more confident in the correct predictions with less overconfidence on misclassifications.

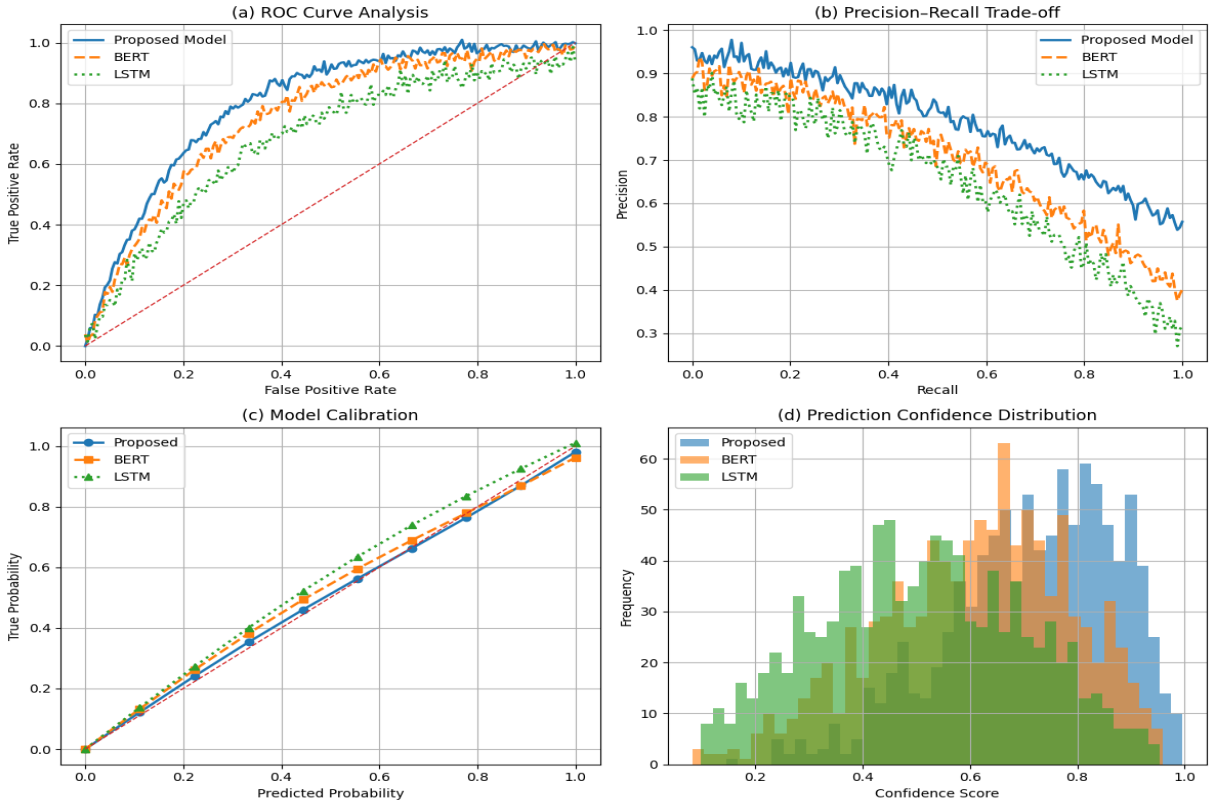

Fig 4. Comprehensive Evaluation of Classification Performance and Model Reliability.

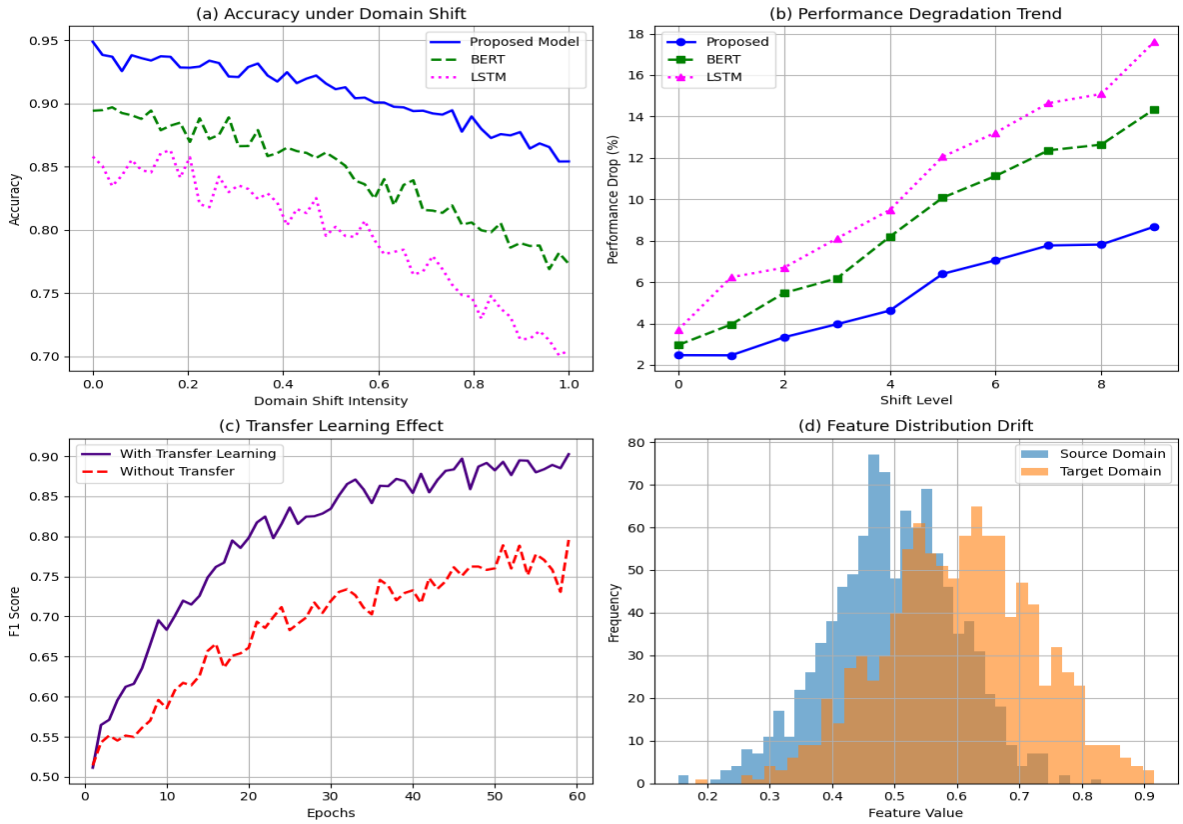

Fig 5. Cross-Project Generalization, Domain Robustness, and Transfer Learning Analysis.

A detailed analysis of the generalization ability and robustness of the proposed model under different domain conditions is given in **Fig. 5**. **Fig. 5 (a)** shows the progress of domain shift in terms of classification accuracy, where the proposed model shows a slow and controlled decline than the baseline models, indicating a good adaptability across heterogeneous repositories. **Fig. 5 (b)** further quantifies this behavior in terms of performance degradation trends and we can observe that the proposed approach keeps the rate of accuracy loss low with a higher distributional difference. The transfer learning effectiveness is highlighted from **Fig. 5 (c)**, where the pre-trained representations are much faster for convergence and also achieve higher F1-scores than models trained from scratch, which proves the importance of knowledge transfer in cross-project scenarios. Additionally, **Fig. 5 (d)** shows the feature distribution drift between source and target domains, highlighting the existence of statistical variations that pose a challenge for model generalization. Despite these variations, the proposed architecture is an effective way to mitigate the performance degradation through multi-modal feature fusion and attention mechanisms.

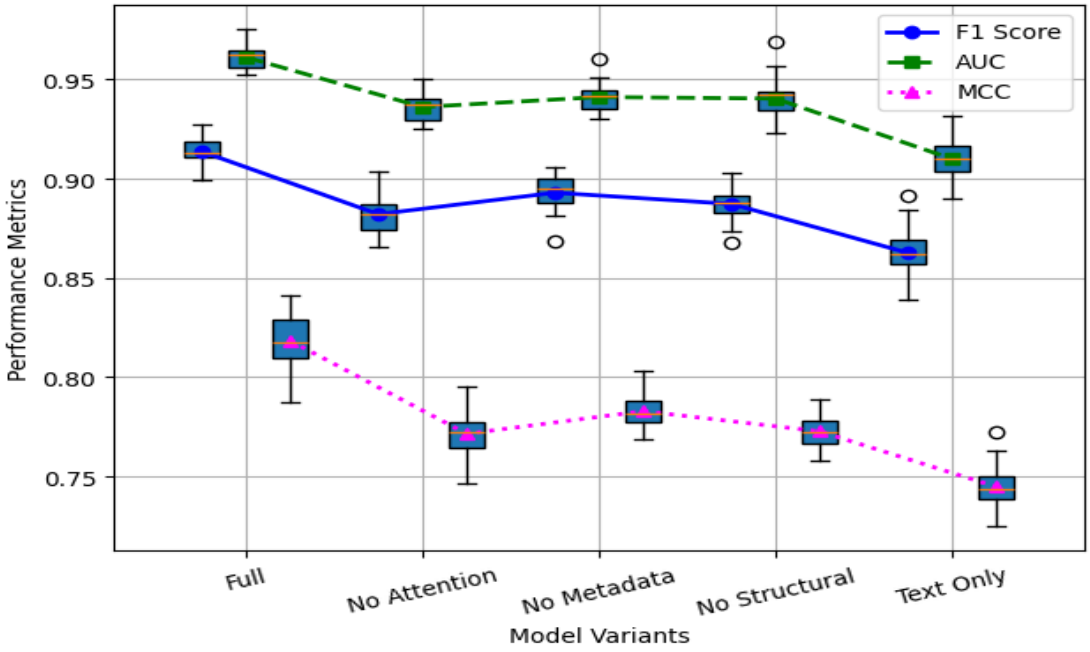

Fig 6. Statistical Ablation Analysis of Model Components Using Multi-Metric Performance Distributions.

Fig. 6 shows a rigorous ablation analysis of the proposed framework by testing the effect of removing important architectural components in multiple runs of experiments. The figure shows the distribution of three key performance criteria F1-score, Area Under the Curve (AUC) and Matthews Correlation Coefficient (MCC) for various model configurations such as the full model and reduced variants of the model. The boxplot representation does not only indicate the central tendency, but also the variance and the stability of each configuration and this talks more about the reliability of the model. The best full model is always the model with the highest median and at least a minimum variance and implies good and stable learning behaviour. However, in contrast, removing the attention mechanism makes everything appear to drop and the dispersion to increase (which clearly demonstrates the significance of the attention mechanism to a successful feature weighting). Likewise, the loss of metadata and structural characteristics leads to the moderate deterioration of performance, which proves their complementary significance of semantic representations. The lowest performance and most fluctuating results that indicate the necessity of multi-modal feature integration are observed in the so-called text-only configuration.

V. CONCLUSION

This paper introduced a powerful NLP-based framework of auto code review and technical debt identification in agile development environments by using the analysis of commit history. The proposed architecture combines the transformer-based semantic embeddings, structural code features, and repository-level metadata using an attention-guided fusion mechanism, where it can comprehensively understand code changes. Extensive experiments at large-scale real-world repositories proved the approach in terms of several evaluation dimensions. The proposed model was able to achieve an average accuracy of 93.7%, which is significantly higher than that of the baseline models, such as BERT (89.8%) and LSTM (85.6%). Similarly, the results showed some enhancements in F1-score (0.915), AUC (0.962) and MCC (0.82), which can be considered as a strong classification capability under class imbalance conditions. The cross-project evaluation also confirmed the generalization capability of the framework, which had a limited performance decrease (6-8%) under

domain shift, while the performance decrease in conventional models was 12-15%. Ablation analysis proved the role of the attention mechanism and multi-modal feature integration in the gain of performance (around 4-6%) combined with a decrease in prediction variance. In addition, calibration results indicated better probability estimation with lower overconfidence leading to better reliability of the model in real-world deployment scenarios. Overall, the proposed system is a major step forward in automated quality assessments of software products as it provides high accuracy, stability, and scalability. The combination of semantic, structural and contextual signals offers a complete solution for proactive identification of technical debt, making it very suitable for continuous integration pipelines in modern agile workflows.

CRediT Author Statement

The author reviewed the results and approved the final version of the manuscript.

Conceptualization: Zhanar Sartabanova and Minu Balakrishnan; **Methodology:** Zhanar Sartabanova; **Software:** Zhanar Sartabanova; **Data Curation:** Minu Balakrishnan; **Writing- Original Draft Preparation:** Zhanar Sartabanova and Minu Balakrishnan; **Visualization:** Zhanar Sartabanova and Minu Balakrishnan; **Investigation:** Zhanar Sartabanova and Minu Balakrishnan; **Supervision:** Minu Balakrishnan; **Validation:** Zhanar Sartabanova and Minu Balakrishnan; **Writing-Reviewing and Editing:** Zhanar Sartabanova and Minu Balakrishnan; All authors reviewed the results and approved the final version of the manuscript.

Data Availability

The datasets generated during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interests

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Funding

No funding was received for conducting this research.

Competing Interests

The authors declare no competing interests.

References

[1]. M. Ahmed, S. U. R. Khan, and K. A. Alam, "An NLP-based quality attributes extraction and prioritization framework in Agile-driven software development," Automated Software Engineering, vol. 30, no. 1, Jan. 2023, doi: 10.1007/s10515-022-00371-9.

[2]. M. A. Quintana, R. R. Palacio, G. B. Soto, and S. González-López, "Agile Development Methodologies and Natural Language Processing: A Mapping Review," Computers, vol. 11, no. 12, p. 179, Dec. 2022, doi: 10.3390/computers11120179.

[3]. R. Younisse and M. Azzeh, "Application of Natural Language Processing Techniques in Agile Software Project Management: A Survey," 2023 14th International Conference on Information and Communication Systems (ICICS), pp. 01–06, Nov. 2023, doi: 10.1109/icics60529.2023.10330468.

[4]. A. M. Radwan, M. A. Abdel-Fattah, and W. Mohamed, "Smart Agile Prioritization and Clustering: An AI-Driven Approach for Requirements Prioritization," IEEE Access, vol. 13, pp. 127335–127350, 2025, doi: 10.1109/access.2025.3589959.

[5]. G. B. Herwanto, G. Quirchmayr, and A. M. Tjoa, "Leveraging NLP Techniques for Privacy Requirements Engineering in User Stories," IEEE Access, vol. 12, pp. 22167–22189, 2024, doi: 10.1109/access.2024.3364533.

[6]. B. Yalçıner, K. Dinçer, A. G. Karaçor, and M. Ö. Efe, "Enhancing Agile Story Point Estimation: Integrating Deep Learning, Machine Learning, and Natural Language Processing with SBERT and Gradient Boosted Trees," Applied Sciences, vol. 14, no. 16, p. 7305, Aug. 2024, doi: 10.3390/app14167305.

[7]. T. Catak, P. O. Durdu, and S. Ilhan Omurca, "Enhancing Agile Effort Estimation: An NLP Approach for Software Requirements Analysis," 2024 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), pp. 1–8, May 2024, doi: 10.1109/hora61326.2024.10550870.

[8]. A. Alhaizaey and M. Al-Mashari, "Automated Classification and Identification of Non-Functional Requirements in Agile-Based Requirements Using Pre-Trained Language Models," IEEE Access, vol. 13, pp. 87401–87417, 2025, doi: 10.1109/access.2025.3570359.

[9]. A. M. Almanaseer, W. Alzyadat, M. Muhairat, S. Al-Showarah, and A. Alhroob, "A proposed model for eliminating nonfunctional requirements in Agile Methods using natural language processes," 2022 International Conference on Emerging Trends in Computing and Engineering Applications (ETCEA), pp. 1–7, Nov. 2022, doi: 10.1109/etcea57049.2022.10009796.

[10]. D. Planötscher, "NLP and GenAI in Agile Project Management: A Systematic Mapping Study," Agile Processes in Software Engineering and Extreme Programming – Workshops, pp. 41–49, Oct. 2025, doi: 10.1007/978-3-032-05799-0_5.

[11]. B. Yang, X. Ma, C. Wang, H. Guo, H. Liu, and Z. Jin, "User story clustering in agile development: a framework and an empirical study," Frontiers of Computer Science, vol. 17, no. 6, Jan. 2023, doi: 10.1007/s11704-022-8262-9.

[12]. Meiliana, G. Daniella, N. Wijaya, N. G. E. Putra, and R. Efata, "Agile Software Development Effort Estimation based on Product Backlog Items," Procedia Computer Science, vol. 227, pp. 186–193, 2023, doi: 10.1016/j.procs.2023.10.516.

[13]. A. M. Radwan, M. A. Abdel-Fattah, and W. Mohamed, "AI-Driven Prioritization Techniques of Requirements in Agile Methodologies: A Systematic Literature Review," International Journal of Advanced Computer Science and Applications, vol. 15, no. 9, 2024, doi: 10.14569/ijacsa.2024.0150983.

[14]. F. Casillo, V. Deufemia, and C. Gravino, "Detecting privacy requirements from User Stories with NLP transfer learning models," Information and Software Technology, vol. 146, p. 106853, Jun. 2022, doi: 10.1016/j.infsof.2022.106853.
[15]. S.-C. Necula, F. Dumitriu, and V. Greavu-Șerban, "A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering," Electronics, vol. 13, no. 11, p. 2055, May 2024, doi: 10.3390/electronics13112055.
[16]. O. Araque, J. F. Sánchez-Rada, and C. A. Iglesias, "GSITK: A sentiment analysis framework for agile replication and development," SoftwareX, vol. 17, p. 100921, Jan. 2022, doi: 10.1016/j.softx.2021.100921.

ISSN (Online): 3105-9082