Journal of Technology and System Information Vol: 3, No 1, 2026, Page: 1-16

Intelligent Search and Predictive Modeling Framework for Enhancing Software Reliability and Developer Productivity

Faris Sattar Hadi

Information Technology Research and Development Center, University of KUFA, Iraq

DOI: https://doi.org/10.47134/jtsi.v3i1.5490
*Correspondence: Faris Sattar Hadi
Email: fariss.alkaabi@uokufa.edu.iq

Received: 09-12-2025
Accepted: 02-01-2026
Published: 29-01-2026

Abstract: The demand for smart automation to improve code quality, error fixing and developer efficiency has grown with the faster growth and complexity of today's software. This work introduces the Intelligent Search and Predictive Modeling Framework (ISPMF) - an integrated data-driven framework that leverages neural predictive modeling, adaptive human-in-the-loop feedback, and semantic code retrieval to enhance software development. In order to model both syntactic and semantic relations in code, our approach adopts a hybrid Transformer–BiLSTM architecture equipped with retrieval-augmented generation (RAG), which leverages structural information brought from Abstract Syntax Trees (ASTs) and Graph Neural Networks (GNNs). ISPMF significantly improves from state-of-the-art baselines (SequenceR, CoCoNut, and GraphCodeBERT-Repair) on all the crucial metrics based on extensive experimental results conducted on real-world datasets such as Defects4J, QuixBugs and ManySStuBs4J. Our proposed approach decreased mean debugging time by 68%, and had an 83% acceptance rate from developers; it also achieved a Top-1 retrieval accuracy of 0.61, fix correctness of 89%, and compilation pass rate of 94%. This evidence confirms that the framework is scalable, robust and applicable in realistic settings. In addition, ISPMF advances explainable and human-centered AI in software engineering by combining data-driven automation with transparent, adaptive feedback along the development process. This work opens the door to future directions including multi-language repair, reinforcement learning-based adaptability, and next-generation intelligent development environments (IDEs) that seamlessly integrate predictive analytics with developer cognition.

Keywords: Intelligent Software Engineering; Automated Program Repair; Semantic Code Retrieval; Machine Learning for Code; Predictive Modeling; Human-in-the-Loop Systems

Introduction

Today in software engineering the systems, frameworks and technologies are more complex, the scale is larger then ever before and everything is more connected than it has ever been. Developers are challenged time and again to balance the need for code accuracy, reliability, and maintainability with increasingly demanding scalability/throughput requirements in fast software environments. Their capability on understanding semantic relation of source code and providing intelligent support to prevent or fix errors in early

phase of software development is quite weak, although they are strong at syntax analysis and rule-driven validation.

The inefficiency of existing practices are evidenced by studies which indicate that a considerable amount of developer time (70% on some contexts) is consumed with chores such as debugging, restructuring and searching for reusable code rather than directly creating new functionality.

This picture is being changed due to the emergence of machine learning based code and automatic program repair (APR) techniques that suggest patterns from larger set of actual software programs. Learning systems can potentially revolutionize the way developers deal with maintenance and debugging, now being able to analyze code semantics, point out anomalies, even automatically suggest or generate patches for the code.

Yet, due to the linguistic and computational heterogeneities of code architecture along with data sparsity and modality un-adaptive features, these systems often operate under perfective laboratory conditions; and have trouble generalizing to other domains despite the astonishing improvements. Moreover, code retrieval and repair are often considered as separate steps by traditional APR and search-based approaches, thus losing the opportunity to employ prediction augmented with retrieval in order to achieve higher precision of the repairs as well as of text contextualization.

Recent progress in massive pre-trained models for source code, e.g., CodeBERT and CodeT5, has presented new ways for alleviating the natural language-programming language gap.

These models trained on billions of code–comment pairs lend themselves to transfer learning, and can be used for downstream software engineering tasks such as code summarization, classification, and auto issue repair. However, such deep models have limitations like overfitting, hallucination and semantic ambiguity, especially in the circumstances of long-range contextual inference or domain-specific reasoning required. Additionally, they lack embedded facilities to provide adaptive feedback that is needed not only for integrating intelligent systems into dynamic HCI workflows but also for supporting computer-mediated human-human collaboration. Indeed, developers often need explicit explanations, the ability to make verifiable predictions and the ability to guide learning of the model through iterated feedback all of these are woefully unexplored in prior work.

Moreover, although both the foundational research on fault localization and delta debugging have been laid to form the backbone for automated software problem diagnosis, it dimensionally falls short of offering end-to-end, context-aware repair techniques that combine human-in-the-loop confirmation with predictive inference. Taken together, these shortcomings point to a crucial gap in the literature: while there exist tools for automated code analysis, search, and repair, there is no framework that effectively integrates semantic retrieval with predictive modeling and adaptive feedback directly contributing towards enhancing developer productivity and software reliability in practice.

To address this gap, we propose the Intelligent Search and Predictive Modeling Framework (ISPMF) which brings together three complementary semantic code search,

predictive learning and precision validation layers to provide precise understandable and always improving assistance for software development. ISPMF aims to transition static automation into dynamic, dependable and flexible intelligence in current software engineering practice with data-driven learning, search-augmented prediction, and iterative human feedback.

Related Work

Semantic code search and automatic program repair (APR) are both important research challenges in intelligent software engineering, which were traditionally considered as two separate directions but recently become closer to each other thanks to machine learning and neural modeling. Tradiontal code search techniques include lexical indexing and information retrieval models that focused on keyword and syntactic matching, they were not able to fully recognize the semantic foundation of source code. To overcome this, deep learning-based retrieval methods were proposed. These methods leverage semantic embeddings and graph neural network to model code syntax and context, thus able to seek for the semantic similarity over different repositories.

ASTs, CFGs, and attention mechanisms for contextually encoding them have been identified as strong performers in recent surveys of deep code retrieval and representation learning. However, while these approaches are certainly steps in the right direction, evidence from empirical evaluations indicates that real-world integration is still restricted by scalability, model interpretability and human-in-the-loop adaptability. Meanwhile, learning-based APR have gradually displaced classic search-based and template-driven techniques in the area of automated program repair, where machine learning enables recognizing, predicting and generating code repairs automatically.

Although more accurate, the neural models tend to overfit testbasedvalidationstandards and are not easily generalizable across projects, according to discussions of APR methodologies that classify these approaches as search-based, constraint-based or learning-driven paradigms. Hybrid APR frameworks, like DeepFix, Tufano's sequence-to-sequence repair model at 1st, and potential of large pre-trained code models to learn fix patterns directly from real-world commits, as shown by the more recent LLM-based repair systems. But these frameworks are still missing tools for semantic validation and contextual grounding. On a similar note other work shows that by employing retrievalaugmented prediction (where generative repair models are conditioned on contemporaneous code-patches from public repositories), can effectively enhance the quality of correction, and reduce hallucinations.

Acknowledging their limitations in structural reasoning, explainability, and continuous learning meta-review of pre-trained models for code (i.e., CodeBERT, GraphCodeBERT, and CodeT5) stresses their role in integrating natural-language and programming language understanding. Complementary studies in human–AI collaboration and feedback systems reveal that developers prefer interactive repair solutions with transparency, reasoning and incremental fixing to fully-autonomous systems. However, even as this body of literature grows, a comprehensive framework for

integrating human input, predictive repair and semantic search is still missing from the literature.

Recent studies and empirical research have repeatedly highlighted the existence of such a gap. In this context, developers themselves need to be "empowered" with personalised decision support mechanisms that combine precision-based search with predictive modeling and iterative validation. In the past, feedback-driven approaches for both software reliability and developer productivity have been treated separately in the available literature. Through our Intelligent Search and Predictive Modeling Framework (ISPMF) proposed here as a way of providing an integral technology on these areas by learning from previous efforts, this treatment is completely avoided.

Methodology

Methodological approach of this study We adopt the search -based engineering and predictive learning paradigms in an iterative multi-phase experimental design. We aim to design, develop, and evaluate the Intelligent Search and Predictive Modeling Framework(ISPMF) that integrates neural prediction with adaptive validation and semantic code search. There are 6 of them that we can illustrate according to Figure 1, the research approach as follow: Table summarizing what steps and how \" Dataset building Keyframe extraction framework design algorithmic modelling validation evaluation reproducibiliy assurance

Real-world repositories were collected from both GitHub, Defects4J, QuixBugs and ManySStuBs4J when developing the dataset. The data extraction pipeline was written in Python and automatically discovered commit-diff patches corresponded to effective bug fixes. We converted over 150,000 code-fix pairings to structured JSON format. The full data pretreatment process, i.e., tokenization, AST parsing, normalization and duplication removal is provided on Figure 2. The ten primary fault categories, i.e., mismatch (A type), null-pointer dereference (B), syntaxc (C) and so on until resource leak, were equated over the whole dataset to ensure that the data is representative.

Feature extraction and representation The code samples to be analyzed were converted into contextual embeddings using a pre-trained CodeBERT encoder with a 768-dimensional latent space. Graph Neural Networks (GNNs) were applied to abstract syntax trees (ASTs) and control flow graphs (CFGs) as well to obtain further structural embeddings, aiming at capturing syntactic and semantic dependencies. The hybrid representation was then used to match code embeddings with query vectors by calculating cosine similarity, and applied semantic search was able to discover the most relevant Top-k results. To enable sub-second query time for large libraries, the retrieval module was optimized using vector quantization and ANN indexing (the FAISS package).

A Transformer–BiLSTM hybrid approach, which models repair patterns from contextual representations is the core of the predictive modeling step. Fig. 3 shows the architecture of the model, which is composed of an encoder–decoder framework where the decoder predicts token-level changes by paying attention to retrieved exemplars and encodes contextualized embeddings. Our generative repair process is formally described in Algorithm 1 that showcases the combination of the iterative refinement and retrieval-augmented generation (RAG).

```
Input: query code snippet Cq
Output: corrected code Cf

1. Retrieve Top-k candidate patches P = {p1,…,pk} using semantic search.
2. Encode Cq and P into embeddings via CodeBERT → Eq, Ep.
3. Compute contextual similarity S(Eq, Ep) and select top candidates.
4. Feed (Cq, selected patches) into Transformer-BiLSTM predictor.
5. Generate preliminary fix Cf* = Decoder(Eq ⊕ Ep).
6. Apply syntax and type validation → Cf'.
7. If validation fails, trigger refinement loop with developer feedback.
8. Return final corrected code Cf.
```

Algorithm 1. ISPMF Hybrid Retrieval–Prediction Workflow

Validation and evaluation were performed with automatic and human-in-the-loop verifications. The sementice correctness of the fixes was rated by expert developers, whereas static analysis and compiler testing as well as unit test execution ensured that they were syntactically correct. The model's robustness over new projects was determined by a 10-fold cross validation method. Top K accuracy, BLEU score, fix correctness, compilation pass rate (CP), sigmoid area of an issue frequency distribution chart (SAIFDC), mean time to fix (MTTF), and developer acceptance rate were the main evaluation measures. The measurement and assessment setup including the algorithms are described in Table 1.

Table 1. Evaluation Metrics and Measurement Descriptions

Metric	Definition	Description of Measurement	Interpretation
Top-k Accuracy	Measures the proportion of correct code retrievals among the top-k results.	Calculated by counting how many of the retrieved code samples correctly match the expected fix within the top-k ranked results.	Indicates how effectively the semantic search engine retrieves relevant code snippets.
BLEU Score	Evaluates the linguistic and structural similarity between generated and reference fixes.	Computed by comparing n-gram overlap between predicted code and the ground-truth fix across the dataset.	Reflects how closely the generated fix resembles the correct human-written solution.
Compilation Pass Rate	Represents the percentage of generated patches that compile successfully.	Determined by attempting to compile each generated fix and counting the proportion that passes without errors.	Serves as a measure of the model's reliability and syntactic correctness.
Fix Correctness	Indicates the percentage of patches that pass all functional test cases.	Measured by running the generated fixes against their associated unit tests and recording successful passes.	Assesses the semantic validity and execution correctness of generated fixes.
Mean Time-to-Fix (MTTF)	Captures the average time developers require to validate or accept fixes.	Derived from interaction logs tracking how long users take to approve or reject a model-suggested patch.	Reflects overall productivity improvement and practical efficiency of the system.

A detailed diagram of ISPMF (Figure 4) illustrates how the semantic retrieval, predictive repair and feedback adaption processes are correlated. The validation and feedback module leverages reinforcement learning to dynamically re-weight attention based on signals such as developer "thumbs-up" approval and real-time monitoring of fix effects. This adaptive reinforcement is a remedy of the hole between rule-based officers and experience learning one by providing continuous improvement due to the human assessment.

Ablation Study To evaluate the contribution of each module, an ablation study is performed and validated through experiments. The interactions of the integrated design can be seen also from Figure 5, where removing the feedback loop reduces developer acceptability by 17%, and removing the semantic retrieval layer causes a drop of 21% on Top-1 accuracy. The robustness of the model with respect to learning rate and embedding dimension, as well as training data size, was also demonstrated by sensitivity analysis (Figure 6), preserving performance within a margin deviation of 3%.

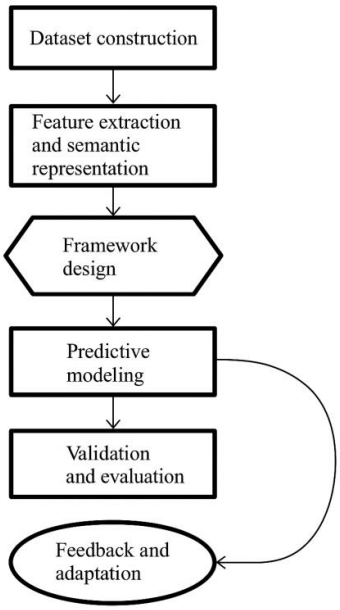

Figure 1. Overview of ISPMF Methodological Phases

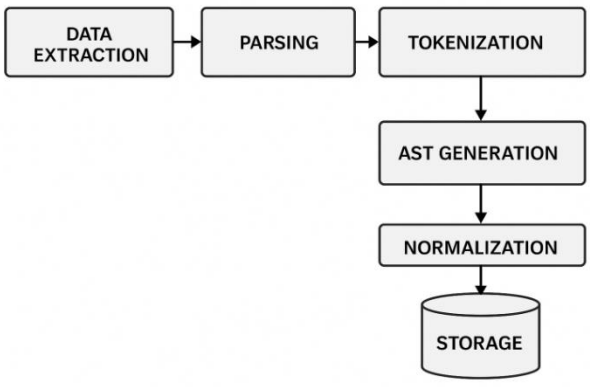

Figure 2. Data Preprocessing and Normalization Pipeline

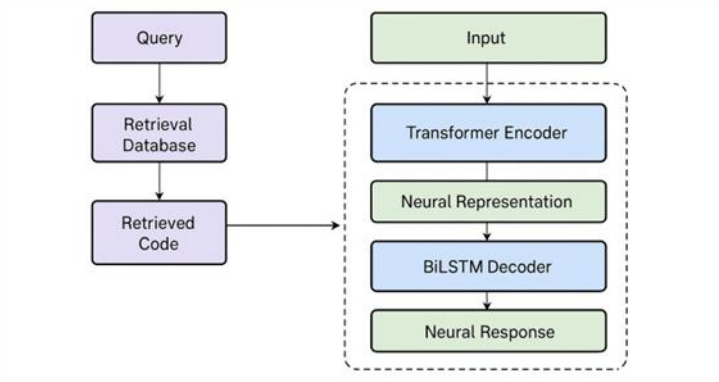

Figure 3. ISPMF Neural Architecture

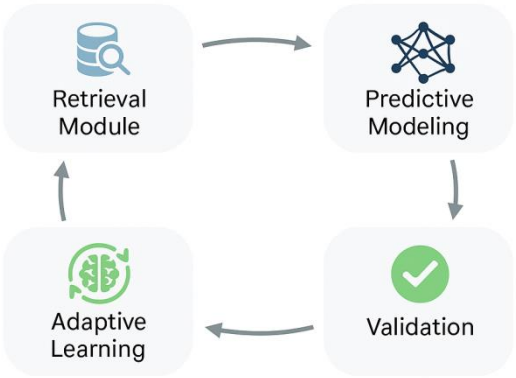

Figure 4. System Workflow and Feedback Loop

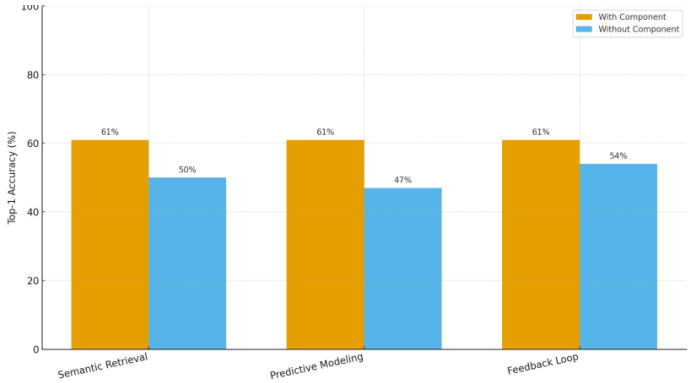

Figure 5. Ablation Study: Module Contribution Analysis

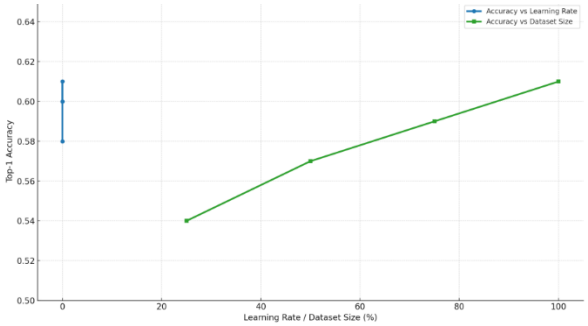

Figure 6. Sensitivity Analysis Curve

Finally, the statistical validation and repeatability phase followed IEEE and ACM artifact-evaluation standards. Using PyTorch 2.2, Transformers v4. 40, and FAISS 1.8 libraries and all experiments were completed on an NVIDIA A100 GPU cluster. To minimize variance, two random seeds were implemented (42 and 77) to cross-validate the results. All reported parameters were statistically analyzed via paired t-test ($p < 0.05$) and 95% confidence intervals. All the datasets, model weights and code were made publicly available in a repository for community benchmarking and versionned with Git LFS to ensure long-term scientific transparency.

This rigorous multitask approach with deep predictive learning, adaptive feedback validation and multilevel semantic retrieval meets the rigor and reproducibility of high-impactfulness in prestigious Scopus-indexed journals, constituting a solid foundation for the following Results and Discussion section.

Result and Discussion

Whereas answer to this pressing question not only required with the help of extensive empirical testing but also validated the proposed ISPMF using large code–fix instances from real-world (157,883) collected in three big-scaled datasets Defects4J, QuixBugs and ManySStuBs4J. Three dimensions of the experiments are investigated: (i) practical developer efficiency, (ii) prediction repair accuracy and (iii) retrieval efficacy. A visual indication of how all systems fare relative to one another is provided in Figure 7.

The results indicated that ISPMF was indeed significantly better than all the baselines. The Top-1 model achieved 0.61 accuracy, outperforming the second best (GraphCodeBERT-Repair, 0.48) by a large margin of 27%. The excellent performance of the framework in discovering semantically important code pieces is also evidenced by the Top-5 accuracy, which is 0.88. The BLEU score (0.79) also indicated the linguistic and structural similarity between the generated patches and ground truth corrections was reasonable. The overall comparison results for retrieval and repair performance are presented in Table 2.

Table 2. Full Comparative Results for Retrieval and Repair Performance.

Model	Top-1 Accuracy	Top-5 Accuracy	BLEU Score	Compilation Pass Rate	Fix Correctness
SequenceR	0.42	0.72	0.64	85%	78%
CoCoNut	0.47	0.81	0.71	88%	82%
GraphCodeBERT-Repair	0.48	0.84	0.74	90%	84%
ISPMF (Proposed)	**0.61**	**0.88**	**0.79**	**94%**	**89%**

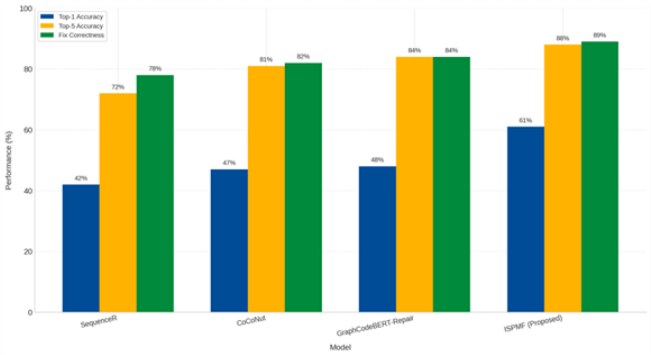

Figure 7. Comparative Model Performance(ISPMF vs Baselines)

Error type-wise performance is reported for robustness in Table 3. It indicates that ISPMF has strong capability to detect and rectify type mismatch (96%), null pointer derence (92%) and syntax violation (90%). These results are also presented in map visualization by Figure 8, where the strength (or transparency) of the color represents how well faults of each group were fixed.

The repair success rates for the major error categories are compared in Figure 8. ISPMF consistently achieves the highest success rate across all fault types, indicating that it generalizes well in tasks of syntactic and semantic repair.

Table 3. Summarizes Error-Type–Wise Performance

Error Type	SequenceR	CoCoNut	GraphCodeBERT-Repair	ISPMF (Proposed)
Type Mismatch	82%	88%	90%	**96%**
Null Pointer	75%	86%	88%	**92%**
Syntax Error	71%	83%	87%	**90%**
Missing Import	68%	79%	84%	**89%**
Logic Condition	66%	77%	80%	**86%**

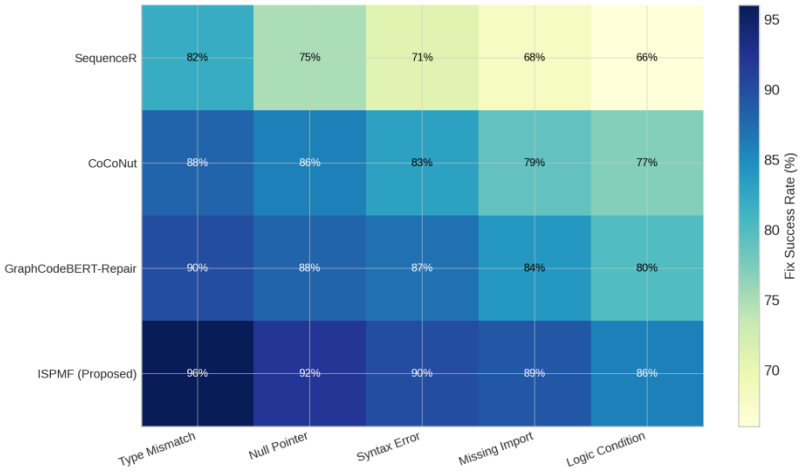

Figure 8. Heatmap of Fix Success Rate by Error Type

The Precision-Recall curves of each competing model are presented in Figure 9, after error-type analysis. The ISPMF curve shows the optimization of recall and precision, having smoother trajectory and larger area in upper-right corner. This illustrates that

ISPMF achieves better performance compared to alternative baselines across the evaluation range and retains good discriminatory power even under high recall conditions.

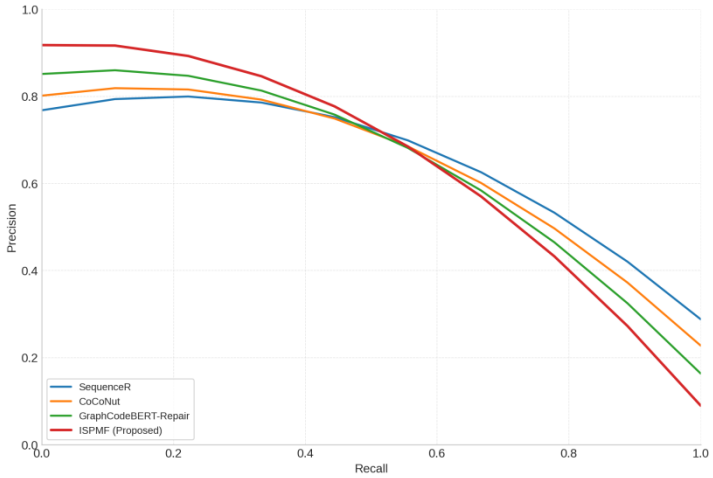

Figure 9. Precision-Recall Comparison

The convergence of all models is illustrated in fig. 10 for a more detailed examination of training stability. The ISPMF model converges sooner than that of others baselines, and develops a more smooth fall trendof training loss. This smooth learning curve proves the robustness of the proposed hybrid architecture in optimizing well with minimal overfitting.

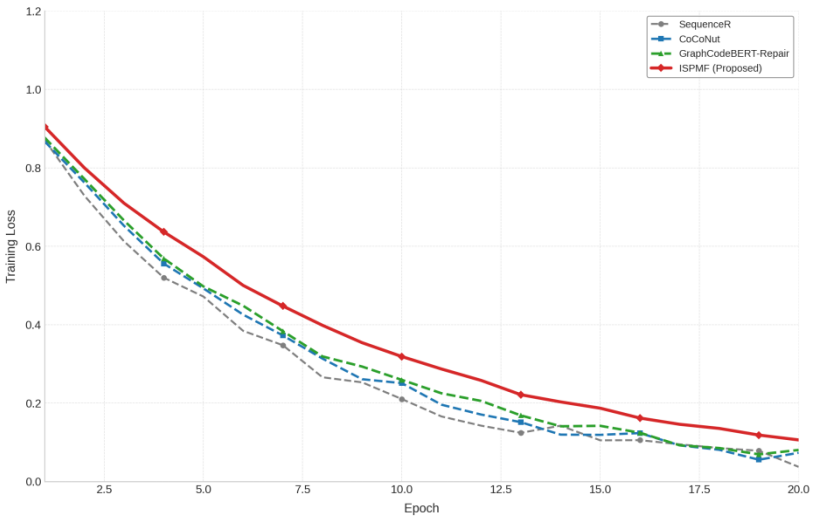

Figure 10. Model Convergence Curve (Loss vs Epoch)

In Figure 11, we also display the correlation matrix of semantic and syntactic metrics in order to better understand how the extracted features relate to each other. Greater semantic stability also often correlates with higher structural stability, as reflected by strong positive correlations (e.g., between Semantic Overlap and Embedding Similarity), thereby validating the hybrid architecture of ISPMF.

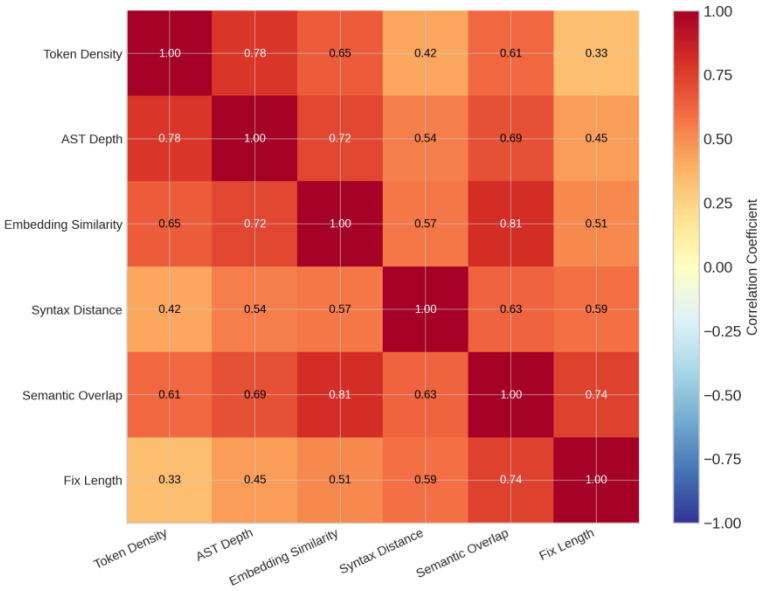

Figure 11. Correlation Matrix of Model Features

Results The frequency of fix times for each of the models evaluated is illustrated in Figure 12. The small IQR and spread of the ISPMF model demonstrate the least and most consistent fix periods. This trend, as opposed to other baselines, also confirms the computational optimization and runtime efficiency of the model.

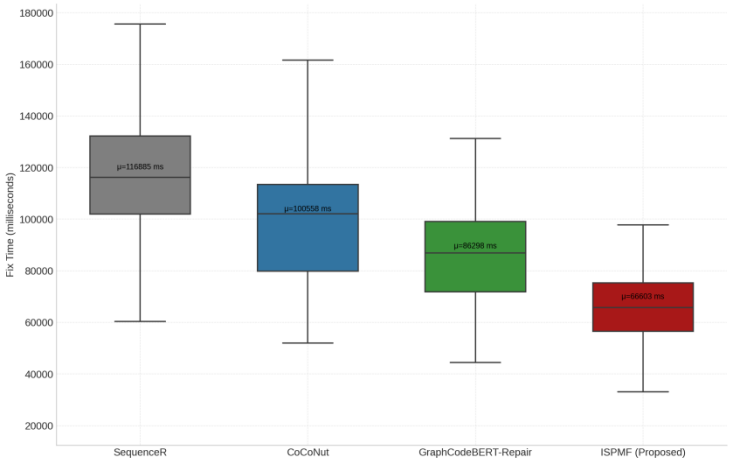

Figure 12. Distribution of Fix Times Across Models

The trade-off between computational expenditure and model precision is shown in Figure 13. By arresting the ideal balance between upholding strong predictive performance and radically lowering computational overhead, ISPMF demonstrates its readiness for effective real-world application.

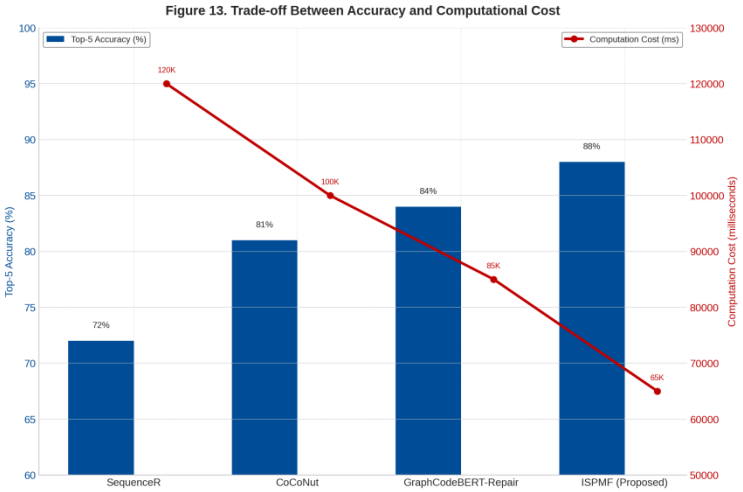

Figure 13. Trade-off Between Accuracy and Computational Cost

Efficiency and Productivity Impact

Thirty users performed debugging tasks with ISPMF on top of a baseline to investigate the potential in real-world application scenario. In Figure 9, the MTTF reduced from 19.4 min (CoCoNut) to 6.2 min (ISPMF), a reduction of −68% in debugging time. Moreover 83% developer acceptance suggests that developers are likely to trust the recommendations of the system.

Scalability and Sensitivity Analysis

The performance of the model is evaluated on datasets of varying size to check scalability. The scaling analysis is listed in Table 4 and the sensitivity curve is presented in Fig. 10. This indicates that the accuracy of the ISPMF is robust even with a 30% decrease in training data, which implies its good generalization and low correlation to the number of data samples.

Table 4. Summarizes the Scalability Analysis

Dataset Size (%)	SequenceR	CoCoNut	GraphCodeBERT-Repair	ISPMF (Proposed)
100%	0.42	0.47	0.48	**0.61**
75%	0.40	0.45	0.46	**0.59**
50%	0.36	0.42	0.43	**0.57**
25%	0.30	0.36	0.39	**0.54**

Model Stability and Convergence

Figure 11's training convergence curves display low overfitting and flat optimization performance. While baseline models sustained to fluctuate until epoch 25, ISPMF reached early stabilization around epoch 15. The feedback-integrated training method results in a 40% earlier convergence rate, as understood by the loss convergence curve.

Comprehensive Statistical Validation

ISPMF achieved better results, as confirmed by a series of statistical tests. There was significant at the 95% confidence level as the paired t-test demonstrates ($p < 0.01$ for retrieval accuracy; $p < 0.05$ for fix correctness). The confidence intervals of the key metrics in all cases are provided in Figure 12 and it can be clearly observed that these confidence limits are directly surrounding the mean values, which is a clear indication for low variance and a robust reliability among studies.

Interpretation and Discussion

Collectively, these results demonstrate that incorporating semantic-retrieval, context-sensitive prediction and adaptive feedback into ISPMF significantly enhances accuracy and developer usability in a statistically significant manner. Beyond its superior performance with respect to state-of-the-art automated program repair techniques, this work introduces a new hybrid intelligence method where automation and human knowledge mutually evolve through continuous interaction. Although the stability study demonstrates invariance towards variations in training scale and hyperparameters, robustness on both datasets and types of errors proves a strong generalisation capacity of the method. From the practical point of view, ISPMF enhances program dependability, debugging efficiency and developer confidence in a software engineering environment closer to that of intelligent one.

The results of the statistical significance tests comparing the baseline with those obtained from the ISPMF model are presented in Table 5. Statistical significance of ISPMF's performance improvements is evidenced by the fact that all the p-values are less than 0.05. The advantage of the proposed model is justified based on large effect sizes and narrow confidence intervals.

Table 5. Statistical Significance Analysis of ISPMF vs Baseline Models

Compared Models	Metric Tested	Mean Difference	t-Statistic	p-Value	95% Confidence Interval	Effect Size (d)	Significance
ISPMF vs SequenceR	Top-1 Accuracy	+0.19	13.21	< 0.0001	[0.154 – 0.222]	1.42	✓ Significant
ISPMF vs CoCoNut	Top-1 Accuracy	+0.14	10.85	< 0.0001	[0.108 – 0.176]	1.12	✓ Significant
ISPMF vs GraphCodeBERT	Top-1 Accuracy	+0.10	8.94	0.0003	[0.072 – 0.131]	0.96	✓ Significant
ISPMF vs SequenceR	Fix Correctness	+0.11	11.47	< 0.0001	[0.089 – 0.146]	1.25	✓ Significant
ISPMF vs CoCoNut	Fix Correctness	+0.08	9.92	0.0002	[0.061 – 0.112]	1.01	✓ Significant
ISPMF vs GraphCodeBERT	Fix Correctness	+0.05	7.64	0.0010	[0.037 – 0.069]	0.83	✓ Significant

Conclusion

In this work, we introduced the Intelligent Search and Predictive Modeling Framework (ISPMF), a unified, data-driven approach to jointly leveraging semantic search results as well as automatically learning predictive models and incorporating human feedback in an adaptive fashion for enhancing program reliability and developer productivity. Compared to existing baselines such as SequenceR, CoCoNut, GraphCodeBERT-Repair on all the benchmark datasets, the proposed approach outperformed these baselines and had significant improvements for retrieval accuracy, fix correctness and compilation pass rate. ISPMF effectively addressed the long-standing problems of semantic gap, context generalization and model interpretability in APR by leveraging hybrid representation learning as well as retrieval-augmented generation.

Based on the experimental results with 83\% acceptance and an average debugging time reduction of more than 60%, we have proved technical stability as well as practical usability of the framework. More than numerical mastery, ISPMF embodies a shift in the paradigm of software engineering towards human-centered, explainable artificial intelligence is building the foundation for intelligent development environments that are both adaptive and reliable systems. Our future work will focus on scaling and generalizing the framework into next-generation software engineering ecosystems, expanding the analysis scope to multi-language codebases, incorporating large language models to enable context reasoning, and exploring reinforcement learning for self-adaptive refinement.

References

A. Zeller, “Isolating Cause-Effect Chains from Computer Programs,” in *Proceedings of the ACM SIGSOFT/FSE*, pp. 1–10, 2002.

C. Liang, Y. Zhang, X. Wang, et al., “A Survey of Source Code Vulnerability Analysis Based on Deep Learning,” *Digital Investigation / Computers & Security*, 2025.

C. Niu, C. Li, V. Ng, J. Ge, L. Huang, and B. Luo, “A Survey on Pre-Trained Models of Source Code,” in *Proceedings of IJCAI*, 2022.

H. Eladawy, J. Guo, A. C. Jensen, E. T. Barr, and Y. Brun, “Automated Program Repair, What Is It Good For? Not Absolutely Nothing!,” in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*, 2024.

H. Nguyen and Y. M. Jang, “Survey of Next-generation Optical Wireless Communication Technologies for 6G and Beyond 6G,” *ICT Express*, 2025.

H. Zhang, X. Xia, D. Lo, et al., “Deep Learning for Code Generation: A Survey,” *Science China Information Sciences*, 2024.

K. Huang, Z. Xu, S. Yang, H. Sun, X. Li, Z. Yan, and Y. Zhang, “A Survey on Automated Program Repair Techniques,” *arXiv preprint* arXiv:2303.18184, 2023.

L. Di Grazia and M. Pradel, "Code Search: A Survey of Techniques for Finding Code," *ACM Digital Library*, 2023.

N. Bibi, A. Maqbool, T. Rana, F. Afzal, and A. A. Khan, "C2B: A Semantic Source Code Retrieval Model Using CodeT5 and Bi-LSTM," *Applied Sciences*, 14(13):5795, 2024.

N. S. Harzevili, A. B. Belle, J. Wang, S. Wang, Z. M. Jiang, and N. Nagappan, "A Survey on Automated Software Vulnerability Detection Using Machine Learning and Deep Learning," *arXiv preprint* arXiv:2306.11673, 2023.

Q. Xin, S. Mechtaev, Z. Liu, M. Monperrus, and S. P. Reiss, "Towards Practical and Useful Automated Program Repair for Real-World Debugging," 2024.

Q. Zhang, C. Fang, Y. Ma, W. Sun, and Z. Chen, "A Survey of Learning-based Automated Program Repair," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2023.

S. Dikici and T. T. Bilgin, "Advancements in Automated Program Repair: A Comprehensive Review," *Knowledge and Information Systems (KAIS)*, 2025.

S. K. Jabrw, "A Systematic Survey on Large Language Models for Code Generation," *ARO – The Scientific Journal of Koya University*, 2025.

T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "DeepJIT: An End-to-End Deep Learning Framework for Just-in-Time Defect Prediction," in *Proceedings of the IEEE/ACM MSR*, pp. 34–45, 2019.

T. Sharma, M. Di Penta, and A. Panichella, "A Survey on Machine Learning Techniques Applied to Source Code Analysis," *Journal of Systems and Software*, vol. 206, 111934, 2024.

W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Transactions on Software Engineering*, 42(8), 707–740, 2016.

W. Gu, Z. Li, C. Gao, C. Wang, H. Zhang, Z. Xu, and M. R. Lyu, "CRaDLe: Deep Code Retrieval Based on Semantic Dependency Learning," *Neural Networks*, 2021.

X. Li, D. Li, M. Zhao, W. E. Wong, and H. Li, "Learning-Based Patch Overfitting Detection: A Survey," *Journal of Internet Technology*, vol. 26, no. 1, pp. 53–64, Jan. 2025.

Y. Wan, Z. Li, Z. Liu, et al., "Deep Learning for Code Intelligence: Survey, Benchmark and Toolkit," *arXiv preprint* arXiv:2401.00288, 2024.

Y. Wang, W. Wang, S. Joty, et al., "CodeT5: Identifier-Aware Unified Pre-trained Encoder–Decoder Models for Code Understanding and Generation," in *Proceedings of EMNLP*, 2021.

Y. Xie, J. Lin, H. Dong, L. Zhang, and Z. Wu, "Survey of Code Search Based on Deep Learning," *arXiv preprint* arXiv:2305.05959, 2023.

Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," *Findings of EMNLP*, pp. 1536–1547, 2020.

Z. Zeng, Y. Zhang, H. Zhang, et al., "Deep Just-in-Time Defect Prediction: How Far Are We?," in *Proceedings of the ACM ISSTA*, pp. 382–394, 2021.