

Intelligent Computing Solutions for Modern Software Challenges

Joelle Pineau

Professor, McGill University, Canada

Abstract

The rapid evolution of digital technologies has transformed software systems into complex, interconnected, data-intensive, and continuously evolving computational environments. Modern software applications are expected to provide high performance, scalability, security, reliability, adaptability, and intelligent decision support while operating across cloud, mobile, edge, and distributed infrastructures. Conventional software engineering approaches remain fundamental, but increasing system complexity has created challenges that require more adaptive and intelligent computational techniques. Intelligent computing integrates artificial intelligence, machine learning, deep learning, natural language processing, evolutionary computation, automated reasoning, predictive analytics, and intelligent optimization to address software development and operational challenges.

This study examines intelligent computing solutions for modern software challenges and proposes an integrated conceptual framework connecting intelligent requirements analysis, software design, automated development, predictive testing, performance optimization, security monitoring, and adaptive maintenance. A conceptual and descriptive research design was adopted through thematic analysis of scholarly literature concerning artificial intelligence in software engineering, machine learning systems, software quality, automated testing, requirements engineering, intelligent optimization, software maintenance, and AI governance. Existing research indicates that intelligent technologies can support multiple software lifecycle activities, including requirements analysis, code generation, defect prediction, test generation, system optimization, and maintenance.

However, their implementation introduces additional concerns involving data quality, model reliability, explainability, security, bias, privacy, technical debt, integration complexity, and human oversight. The proposed framework positions intelligent computing as an augmentation mechanism rather than a complete replacement for software engineering expertise. The framework consists of intelligent data acquisition, requirement interpretation, adaptive architecture selection, automated development support, predictive quality assurance, intelligent deployment, continuous monitoring, and feedback-based optimization.

The study further evaluates the role of machine learning and generative artificial intelligence in improving software productivity and system performance while recognizing the need for validation and governance. The analysis suggests that intelligent computing can contribute to faster development cycles, improved defect identification, adaptive resource management, and more responsive software maintenance when implemented within structured engineering processes. The study concludes that successful intelligent software development requires a balanced combination of computational

intelligence, engineering discipline, human expertise, security controls, and organizational governance. Future research should investigate autonomous software engineering, explainable intelligent development tools, AI-assisted software architecture, trustworthy agentic systems, adaptive quality assurance, privacy-preserving development environments, and human-centered governance frameworks.

Keywords: Intelligent Computing, Software Engineering, Artificial Intelligence, Machine Learning, Software Optimization, Automated Testing, Predictive Analytics, Software Quality, Intelligent Systems, Digital Transformation.

1. Introduction

The contemporary software landscape is characterized by unprecedented complexity, rapid technological change, and continuously increasing user expectations. Software applications now operate across cloud platforms, mobile devices, edge infrastructures, distributed networks, enterprise environments, and highly interconnected digital ecosystems. Organizations expect software systems to deliver high availability, rapid response, scalability, security, usability, and adaptability while simultaneously reducing development and maintenance costs. These requirements create significant pressure on conventional software engineering practices. As software systems become larger and more interconnected, manual analysis, testing, optimization, and maintenance become increasingly difficult. Intelligent computing has consequently emerged as an important technological approach for addressing complex software challenges through computational learning, automated reasoning, predictive analysis, and adaptive decision-making.

Artificial intelligence and machine learning are increasingly influencing the software lifecycle. Recent research identifies applications across requirements engineering, design, development, testing, release, maintenance, and software optimization. AI-based approaches can analyze large volumes of development data, identify recurring patterns, predict defects, generate code, recommend design alternatives, automate testing activities, and support software maintenance. A systematic mapping study of software engineering for AI-based systems found that testing and software quality have received substantial research attention, while maintenance and several other areas remain comparatively less developed. Data-related issues were also identified as recurring challenges. More recent research further emphasizes the growing role of generative AI and large language models in software engineering, while highlighting the need for reliable datasets, meaningful benchmarks, digital ethics, and protection of human values.

The concept of intelligent computing extends beyond the application of one artificial intelligence technique. It represents an integrated computational approach in which software systems can interpret information, learn from historical observations, predict future conditions, optimize operations, and adapt to changing environments. Machine learning can support predictive defect analysis, deep learning can identify complex patterns, natural language processing can interpret requirements and documentation, and optimization algorithms can improve resource allocation. Generative AI can assist with code generation, transformation, documentation, testing, and software comprehension. Research published in 2025 indicates that AI applications are increasingly being investigated across software requirements, automated code transformation, predictive modeling, fault monitoring, and project guidance.

Despite these opportunities, intelligent computing introduces a new class of software engineering challenges. Traditional software generally follows explicitly programmed rules, whereas machine learning systems can depend heavily on data, training procedures, statistical behavior, and model configuration. Consequently, software quality can no longer be evaluated exclusively through conventional functional correctness. Developers may also need to consider model robustness, data representativeness, fairness, explainability, uncertainty, drift, and monitoring. Requirements engineering for AI systems presents additional complexity because AI behavior may be probabilistic and difficult to specify completely in advance. Recent systematic research has identified emerging requirements involving data and ethics and has emphasized the limited empirical evaluation of areas such as trust and explainability.

Security and governance represent another important dimension. Intelligent development tools may process source code, software documentation, proprietary information, user data, or operational logs. If these resources are handled without appropriate safeguards, organizations may face confidentiality, privacy, intellectual property, and cybersecurity risks. AI-generated code can also contain defects or vulnerabilities if it is accepted without sufficient review and validation. Recent industry evidence suggests that the rapid adoption of AI agents in software engineering is creating governance and security pressures, reinforcing the importance of visibility, verification, access controls, and operational safeguards.

The purpose of this study is therefore to examine intelligent computing solutions for modern software challenges and develop a comprehensive conceptual framework for their integration throughout the software lifecycle. The study focuses on intelligent requirements analysis, architecture support, automated development, predictive testing, performance optimization, security monitoring, deployment, and adaptive maintenance. It evaluates both the potential benefits and limitations of intelligent computing and emphasizes the importance of combining automation with human expertise and responsible governance.

2. Objectives of the Study

2.1 Primary Objective

The primary objective of this study is to examine intelligent computing solutions for modern software challenges and determine how artificial intelligence, machine learning, predictive analytics, automated reasoning, intelligent optimization, and adaptive monitoring can improve software development, quality, security, performance, maintainability, and lifecycle decision-making across complex digital environments.

2.2 Evaluation of Intelligent Software Technologies

This objective evaluates artificial intelligence, machine learning, deep learning, natural language processing, generative AI, predictive analytics, and optimization techniques according to their usefulness across software development activities. It examines their contributions to requirements analysis, coding, testing, debugging, deployment, performance improvement, security monitoring, and maintenance while considering reliability, explainability, data requirements, integration complexity, and human oversight.

2.3 Development of an Intelligent Software Framework

This objective develops a conceptual framework integrating intelligent requirements analysis, adaptive software architecture, automated development support, predictive quality assurance, intelligent deployment, continuous monitoring, and feedback-based maintenance. The framework aims to establish a coordinated approach in which computational intelligence supports software engineers while maintaining validation, governance, security, transparency, accountability, and organizational control.

3. Review of Literature

3.1 Artificial Intelligence in Software Engineering

Artificial intelligence has increasingly influenced software engineering by supporting activities traditionally dependent on extensive manual effort. Machine learning can identify defect patterns, recommend test cases, classify requirements, and predict maintenance needs. Natural language processing can assist with requirements interpretation and documentation analysis, while intelligent recommendation systems can support software design and development decisions. Research indicates that AI has been investigated across multiple software engineering phases, demonstrating its potential to improve both efficiency and analytical capability.

Systematic research on software engineering for AI-based systems demonstrates that the field has expanded substantially, with software testing and quality receiving significant attention. However, software maintenance remains comparatively underexplored. Researchers also identify data-related challenges as recurring concerns, reflecting the importance of reliable training, validation, and operational datasets. These findings indicate that intelligent software engineering requires specialized processes that combine conventional engineering practices with data management, model evaluation, and continuous monitoring.

3.2 Intelligent Automation and Generative Computing

Generative artificial intelligence has introduced new possibilities for software development by assisting with source-code generation, code explanation, refactoring, documentation, test creation, and software comprehension. Large language models can process natural-language descriptions and generate computational artifacts that may reduce development effort. Recent literature identifies generative AI and large language models as major areas shaping the future of AI-assisted software engineering. However, effective use requires careful validation because generated outputs may contain logical errors, security weaknesses, incorrect assumptions, or inappropriate dependencies.

Intelligent automation can also support software optimization and operational decision-making. Learning-based systems can identify performance bottlenecks, predict system failures, recommend configuration changes, and support resource allocation. Recent research into AI-driven software optimization indicates that large language models and automated techniques are being investigated for improving software performance, although correctness and performance validation remain essential. Intelligent automation therefore creates opportunities for greater efficiency but requires strong testing and verification mechanisms.

3.3 Intelligent Quality Assurance and Adaptive Software Systems

Software quality assurance is one of the areas in which intelligent computing can provide significant benefits. Machine learning can analyze historical defects and identify code or components associated with increased failure probability. Intelligent testing systems can prioritize test cases, generate test scenarios, detect anomalies, and identify regression risks. Such approaches can help testing teams allocate resources to high-risk components and reduce repetitive manual activities.

Adaptive software systems require continuous observation because system behavior and operating conditions can change over time. Intelligent monitoring can identify performance deterioration, security anomalies, unexpected user behavior, and resource consumption patterns. However, adaptive systems also create additional maintenance requirements because models may need retraining, validation, recalibration, or replacement. Research on AI-based software engineering emphasizes that data management, quality, testing, and maintenance remain central challenges for reliable AI-enabled software systems.

4. Research Methodology

4.1 Research Design

This study adopts a conceptual and descriptive research design to examine intelligent computing solutions for modern software challenges. The approach integrates concepts from artificial intelligence, machine learning, software engineering, software architecture, cybersecurity, predictive analytics, optimization, and intelligent information systems to establish a multidisciplinary understanding of modern software development.

A conceptual design was selected because intelligent computing affects multiple software lifecycle stages rather than one isolated development activity. The study therefore focuses on relationships between intelligent technologies, software processes, quality objectives, operational requirements, security controls, and organizational decision-making.

4.2 Data Collection and Thematic Analysis

Secondary scholarly literature concerning AI in software engineering, intelligent development, software testing, requirements engineering, machine learning systems, automated optimization, and software quality was reviewed. The literature was organized around recurring technical and organizational themes.

Thematic analysis was used to identify the principal applications, benefits, limitations, and implementation challenges associated with intelligent computing. Particular attention was given to software productivity, quality assurance, predictive capability, performance optimization, cybersecurity, data dependency, explainability, and human oversight. Recent research confirms that data, testing, quality, ethics, and governance remain important areas of concern.

4.3 Proposed Intelligent Computing Framework

The proposed framework follows the sequence:

Software Requirements → Intelligent Analysis → Adaptive Architecture → AI-Assisted Development → Predictive Testing → Intelligent Deployment → Continuous Monitoring → Adaptive Maintenance

The first stage collects functional, technical, security, performance, and user requirements. Intelligent analysis uses natural language processing and machine learning to classify requirements, identify inconsistencies, and detect missing information.

Adaptive architecture evaluates system characteristics and recommends suitable computational structures. AI-assisted development supports code generation, refactoring, documentation, and development tasks. Predictive testing evaluates software components according to risk and historical defect patterns.

Intelligent deployment monitors operational conditions and resource requirements. Continuous monitoring detects performance degradation, security anomalies, and behavioral changes. Adaptive maintenance uses monitoring feedback to prioritize updates, optimization, testing, and model or component replacement.

5. Analysis

5.1 Intelligent Computing Across the Software Lifecycle

Intelligent computing can influence nearly every major stage of software engineering. During requirements analysis, natural language processing can classify requirements, identify similarities, detect ambiguity, and support traceability. During architecture and design, intelligent recommendation systems can compare architectural alternatives and identify potential risks.

During development, machine learning and generative AI can assist programmers with code generation, code explanation, refactoring, documentation, and error identification. However, generated code must be reviewed because automated generation does not guarantee functional correctness, security, maintainability, or compliance.

During testing, machine learning can prioritize test cases based on historical defect patterns. Predictive models can identify high-risk components and support targeted testing. Intelligent testing can therefore complement rather than replace conventional verification.

Table 1: Intelligent Computing Applications Across Software Lifecycle Stages

Software Lifecycle Stage	Intelligent Computing Technique	Major Function	Expected Benefit
Requirements	NLP and ML	Requirement classification	Improved interpretation
Planning	Predictive Analytics	Effort and risk prediction	Better planning
Architecture	Intelligent Recommendation	Architecture evaluation	Better design decisions
Development	Generative AI	Code assistance	Increased productivity

Software Lifecycle Stage	Intelligent Computing Technique	Major Function	Expected Benefit
Debugging	Machine Learning	Error identification	Faster resolution
Testing	Predictive Analytics	Risk-based testing	Improved test efficiency
Security	Anomaly Detection	Threat identification	Earlier detection
Deployment	Intelligent Optimization	Resource allocation	Improved efficiency
Monitoring	Machine Learning	Behavioral analysis	Faster incident detection
Maintenance	Predictive Models	Failure prediction	Proactive maintenance

5.2 Intelligent Software Performance Optimization

Software performance is increasingly difficult to manage because modern applications depend on distributed services, cloud infrastructure, databases, APIs, containers, and external platforms. Performance bottlenecks can emerge from inefficient algorithms, database queries, memory usage, network communication, resource contention, or poorly configured infrastructure.

Intelligent computing can analyze historical performance data and identify patterns associated with degradation. Machine learning models can predict resource requirements and detect abnormal behavior before failures occur.

Automated optimization can recommend configuration changes, resource scaling, caching strategies, database improvements, or code-level modifications. New research is exploring large language models as software system optimizers, including profiling-guided analysis and automated performance improvements, while emphasizing correctness and performance validation.

Intelligent optimization can also support cloud environments. Instead of maintaining fixed resource allocations, systems can predict demand and dynamically adjust computing capacity. Such approaches may improve resource utilization and reduce unnecessary infrastructure costs.

Table 2: Intelligent Solutions for Major Modern Software Challenges

Modern Software Challenge	Intelligent Solution	Primary Technology	Expected Impact
Requirement Ambiguity	Automated requirement analysis	NLP	Improved clarity
Code Complexity	AI-assisted coding	Generative AI	Reduced development effort
Software Defects	Defect prediction	Machine Learning	Earlier detection
Testing Scale	Intelligent test prioritization	Predictive Analytics	Improved testing efficiency
Performance Bottlenecks	Automated optimization	AI/ML	Better performance
Cybersecurity Threats	Anomaly detection	Machine Learning	Faster threat identification
Resource Waste	Predictive allocation	AI Optimization	Better utilization
Maintenance Burden	Failure prediction	Predictive Analytics	Proactive maintenance
Documentation Complexity	Automated generation	NLP/LLMs	Improved documentation
System Adaptability	Continuous learning	Intelligent Agents	Greater adaptability

5.3 Intelligent Security and Quality Management

Security represents a critical requirement for intelligent software systems. Traditional security mechanisms remain necessary, but intelligent monitoring can provide additional capabilities by identifying unusual activity and learning from historical security events.

Machine learning can analyze system logs, network activity, user behavior, and application events to identify deviations from expected patterns. Intelligent detection can support early identification of suspicious activities. However, machine-learning security systems can generate false positives or fail to detect previously unseen attacks. Consequently, intelligent security mechanisms should operate alongside established security controls rather than replacing them.

Software quality management can similarly benefit from predictive analysis. Historical defect data can identify modules with elevated risk, while intelligent testing can prioritize verification activities. AI-based software engineering research emphasizes the importance of dependable and safe systems, with data-related concerns and quality assurance remaining important areas of investigation.

6. Discussion

Intelligent computing provides a broad set of opportunities for addressing the growing complexity of modern software systems. Its strongest contribution lies in augmenting software engineers with analytical and automation capabilities that would be difficult to perform manually at large scale. Requirement analysis, code assistance, defect prediction, intelligent testing, performance optimization, and operational monitoring can all benefit from computational intelligence. However, the effectiveness of these approaches depends on the quality of data, models, development processes, and validation mechanisms supporting them.

The integration of generative AI represents a particularly significant development. Large language models can assist developers with code generation, documentation, explanation, transformation, and software comprehension. Nevertheless, generated outputs cannot automatically be considered reliable. Developers must validate functionality, security, performance, licensing implications, and maintainability. Recent research identifies generative AI, large language models, data quality, benchmarks, and digital ethics as important future challenges for AI-enabled software engineering.

Intelligent computing should therefore be implemented as a human-centered augmentation strategy rather than an unrestricted replacement for software engineering expertise. Organizations should establish governance mechanisms defining acceptable uses, validation responsibilities, security controls, data-handling requirements, and escalation procedures. Intelligent systems should provide recommendations and automation while allowing qualified professionals to review important decisions. This balanced approach can enable organizations to obtain productivity and optimization benefits while reducing the risks associated with automation, uncertainty, bias, security vulnerabilities, and inappropriate system behavior.

7. Challenges

7.1 Data Quality and Model Reliability

Intelligent software systems depend heavily on the quality, representativeness, and relevance of their training and operational data. Incomplete, inconsistent, biased, or outdated datasets can produce unreliable predictions and inappropriate recommendations. Data quality therefore directly affects the reliability of intelligent development and operational processes.

Model reliability is equally important because software engineers may depend on predictions for testing, maintenance, security, or resource allocation. Models can perform differently across environments and may degrade as software, users, or operational conditions change. Continuous validation is therefore necessary to maintain dependable performance.

7.2 Explainability and Human Oversight

Many intelligent systems produce recommendations that may be difficult for developers or managers to understand. Limited explainability can reduce confidence and make it difficult to determine why a system recommended a particular architecture, generated a specific code structure, or classified a software component as high risk.

Human oversight is therefore essential for high-impact software decisions. Developers should retain the ability to review, reject, modify, and validate automated recommendations. Explainable interfaces and traceable decision processes can improve trust while ensuring that computational intelligence remains accountable to responsible human decision-making.

7.3 Security, Privacy, and Ethical Risks

Intelligent development tools may process source code, requirements, user information, logs, proprietary documentation, and operational data. Improper handling can expose confidential information or create privacy and intellectual-property risks. Organizations must therefore establish secure data-management policies for AI-enabled development environments.

Ethical risks may also arise from biased datasets, unfair automated decisions, insecure generated code, and inappropriate automation. Recent research highlights the need for digital ethics and responsible practices as AI becomes more deeply integrated into software engineering. Security and ethical governance must therefore accompany technical innovation.

7.4 Integration and Organizational Complexity

Integrating intelligent computing into established software development environments can require significant architectural and organizational changes. Organizations may need new data pipelines, AI infrastructure, monitoring tools, model-management systems, security controls, and employee training. Existing development workflows may also require redesign.

Organizational resistance can further slow adoption. Developers may distrust automated tools, while managers may have difficulty measuring the actual benefits of AI-assisted development. Successful implementation therefore requires clear objectives, appropriate training, incremental deployment, measurable performance indicators, and continuous evaluation.

8. Future Research

Future research should investigate autonomous software engineering systems capable of coordinating requirements analysis, coding, testing, deployment, and maintenance. Such systems should combine multiple intelligent agents while maintaining strong verification and governance mechanisms. Research should focus on reliability, transparency, secure agent interaction, human oversight, and controlled autonomy for complex software environments.

Further studies should explore explainable AI for software engineering. Intelligent development tools should provide developers with understandable reasons for recommendations, generated outputs, risk classifications, and optimization decisions. Explainability can improve trust, debugging, accountability, and responsible adoption while helping organizations evaluate whether AI-generated recommendations satisfy technical and organizational requirements.

Future research should also examine intelligent software systems operating across cloud, edge, and distributed environments. Adaptive computing could dynamically select processing locations according to latency, cost, privacy, resource availability, and workload characteristics. Such research may support resilient digital ecosystems capable of continuously adjusting software behavior while maintaining security, performance, and operational reliability.

9. Conclusion

Intelligent computing provides a significant opportunity to address modern software challenges by combining artificial intelligence, machine learning, predictive analytics, natural language processing, optimization, and automated reasoning with established software engineering practices. These technologies can support requirements analysis, development, testing, security, performance management, deployment, and maintenance while reducing repetitive effort and improving analytical decision-making.

The study demonstrates that intelligent software engineering should not be understood simply as automated programming. Effective intelligent systems require reliable data, appropriate models, robust validation, secure infrastructure, continuous monitoring, and responsible human oversight. Without these supporting mechanisms, automation can introduce new risks involving inaccurate outputs, security vulnerabilities, bias, privacy concerns, technical debt, and inappropriate decisions.

Generative AI and intelligent optimization are creating particularly important opportunities for software development. AI-assisted coding, automated documentation, predictive testing, performance optimization, and intelligent monitoring can improve development productivity and operational efficiency. However, generated or automated results must be systematically evaluated because computational intelligence does not eliminate the need for software engineering expertise and quality assurance.

In conclusion, intelligent computing should be implemented as a controlled, human-centered augmentation of software engineering. Organizations should combine intelligent automation with secure development practices, transparent governance, continuous testing, data-quality management, and professional judgment. Future progress will depend on autonomous yet accountable software systems, explainable AI, trustworthy intelligent agents, adaptive architectures, and responsible governance frameworks capable of supporting increasingly complex digital environments.

References

1. Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Computing Surveys*, 51(4), Article 81, 1–37. <https://doi.org/10.1145/3212695>
2. Arpteg, A., Brinne, B., Crnkovic-Friis, L., & Bosch, J. (2018). Software engineering challenges of deep learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 50–59). IEEE. <https://doi.org/10.1109/SEAA.2018.00018>
3. Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1), 1–127. <https://doi.org/10.1561/22000000006>
4. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. <https://doi.org/10.1023/A:1010933404324>
5. Chandra, S., & Saini, S. (2012). Intelligent computing techniques for software engineering applications. *International Journal of Computer Applications*, 48(17), 1–6.
6. Godefroid, P., Peleg, H., & Singh, R. (2017). Learn&Fuzz: Machine learning for input fuzzing. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 50–59). IEEE. <https://doi.org/10.1109/ASE.2017.8115618>

7. Gu, X., Zhang, H., Zhang, D., & Kim, S. (2016). Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 631–642). Association for Computing Machinery. <https://doi.org/10.1145/2950290.2950334>
8. Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.
9. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer. <https://doi.org/10.1007/978-0-387-84858-7>
10. Hellendoorn, V. J., Bird, C., Barr, E. T., & Allamanis, M. (2018). Deep learning type inference. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 152–162). Association for Computing Machinery. <https://doi.org/10.1145/3236024.3236055>
11. Jiang, S., Armaly, A., & McMillan, C. (2017). Automatically generating commit messages from diffs using neural machine translation. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 135–146). IEEE.
12. Li, X., Jiang, H., Ren, Z., Li, G., & Zhang, J. (2018). Deep learning in software engineering. *arXiv*. <https://arxiv.org/abs/1805.04825>
13. Liu, H., Xu, Z., & Zou, Y. (2018). Deep learning based feature envy detection. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 385–396). IEEE.
14. Murphy, C., Kaiser, G., & Chu, W. (2007). Toward a software testing and debugging approach based on machine learning and data mining. In *Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis*. Association for Computing Machinery.
15. Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1), 2–13. <https://doi.org/10.1109/TSE.2007.256942>
16. Poshyvanyk, D., Marcus, A., Ferenc, R., & Vance, R. (2007). Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering*, 12(1), 5–32. <https://doi.org/10.1007/s10664-006-9011-9>
17. White, M., Tufano, M., Vendome, C., & Poshyvanyk, D. (2016). Deep learning code fragments for code clone detection. In *2016 IEEE/ACM 31st International Conference on Automated Software Engineering (ASE)* (pp. 87–98). IEEE.
18. Xu, B., Ye, D., Xing, Z., Xia, X., Chen, G., & Li, S. (2016). Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *2016 IEEE/ACM 31st International Conference on Automated Software Engineering (ASE)* (pp. 51–62). IEEE.
19. Zhao, G., & Huang, J. (2018). DeepSim: Deep learning code functional similarity. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 141–151). Association for Computing Machinery.
20. Zhao, J., Albarghouthi, A., Rastogi, V., Jha, S., & Outeau, D. (2018). Neural-augmented static analysis of Android communication. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 342–353). Association for Computing Machinery.