The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 07-15
DOI 10.37547/tajet/Volume07Issue04-02

Harnessing Graph Neural Networks (Gnn) For Automated Test Case Prioritization: Challenges and Opportunities in Qa Automation

Anatolii Husakovskyi

Master of Science in Systems Programming (Computer Engineering), National Aerospace University "Kharkiv Aviation Institute", Ukraine

OPEN ACCESS

SUBMITTED 23 February 2025
ACCEPTED 25 March 2025
PUBLISHED 03 April 2025
VOLUME Vol.07 Issue04 2025

CITATION

Anatolii Husakovskyi. (2025). Harnessing Graph Neural Networks (Gnn) For Automated Test Case Prioritization: Challenges and Opportunities in Qa Automation. The American Journal of Engineering and Technology, 7(04), 07–15. https://doi.org/10.37547/tajet/Volume07Issue04-02

Abstract: Graph Neural Networks (GNNs) present significant potential to revolutionize automated Test Case Prioritization (TCP) in Quality Assurance (QA) by effectively modeling intricate software-test relationships. This study evaluates the performance of Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT) against traditional prioritization methods, including random, coverage-based, and historical-data-based prioritization. Employing five publicly available software project datasets, results indicate that GNN-based methods, particularly GCN, demonstrate superior performance with an average APFD (Average Percentage Faults Detected) score of 84.2%, outperforming conventional approaches. Despite their effectiveness, GNN methods face substantial challenges, notably computational complexity, scalability issues, data availability and quality concerns, and limited interpretability. Practical adoption also demands sophisticated graph construction, rigorous hyperparameter tuning, and integration into existing QA workflows. The findings emphasize the necessity for strategic implementation and further research in hybrid modeling, incremental learning, and explainable AI to maximize the benefits of GNNs in TCP.

Keywords: Graph Neural Networks (GNN), Test Case Prioritization (TCP), Quality Assurance (QA), Software Testing, Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), APFD, Scalability, Interpretability, Machine Learning in Software Engineering.

Introduction: Software development practices have significantly evolved over recent decades, driven by escalating consumer expectations and rapid technological innovations. In this accelerated development environment, ensuring the delivery of reliable and high-quality software products remains a paramount concern for software organizations worldwide. Quality Assurance (QA) plays a critical role in software engineering by systematically detecting, reporting, and managing software defects. Automation within QA, in particular, has been transformative, allowing software testing teams to conduct extensive tests quickly, efficiently, and consistently, ensuring faster releases and higher product quality.

One vital aspect of automated QA practices is Test Case Prioritization (TCP). TCP aims to arrange test cases in a specific execution order to maximize the likelihood of detecting faults earlier in the testing process (Elbaum et al., 2002). The underlying principle of TCP is based on the recognition that, due to limited resources and tight delivery schedules, running all available test cases during each testing cycle is often impractical or impossible. Hence, prioritizing tests ensures the most critical or fault-prone aspects of the system are tested first, effectively leveraging available resources to maintain software reliability and customer satisfaction.

Various conventional methods have historically dominated TCP, including:

- Coverage-based prioritization: where test cases covering the most extensive or most critical parts of the codebase are executed first.
- Risk-based prioritization: focusing on components or functionalities deemed most likely to contain faults or have the highest potential negative impact.
- Historical-data-based prioritization: using data from previous testing cycles to prioritize test cases known to be effective in identifying faults in past iterations.

Although these conventional methods have shown practical utility, they possess inherent limitations. For example, coverage-based methods, though intuitive, rely heavily on the assumption that code coverage correlates directly to fault detection, which might not always hold true (Hemmati et al., 2015). Risk-based approaches are subjective and heavily reliant on expert judgment, potentially introducing biases or inconsistencies (Catal & Mishra, 2013). Historical-data-based prioritization might fail when applied to new or significantly modified systems with insufficient historical records, limiting its generalizability and applicability.

Recent advancements in machine learning, particularly the emergence of Graph Neural Networks (GNNs), present promising opportunities to overcome the limitations inherent in traditional TCP methods. GNNs have attracted considerable attention due to their exceptional capability to represent and learn complex relationships within structured data, particularly graph-structured data, common in various fields, including social networks, recommender systems, and biological networks (Zhou et al., 2020). Leveraging these strengths, researchers have started exploring the applicability of GNNs in QA automation by treating software components, modules, and test cases as interconnected nodes in a graph structure, explicitly modeling their complex interdependencies.

This novel application of GNNs in TCP stems from the realization that software systems inherently exhibit a graph-like nature, where software modules, components, methods, and test cases have intrinsic relational dependencies, particularly evident through function calls, data flows, and interactions during execution. Such graph structures naturally align with the representational power of GNNs, allowing these neural networks to capture intricate relationships and dynamically evolving dependencies within software systems that conventional prioritization methods fail to account for effectively.

Despite the promising prospects, employing GNN-based methods in automated TCP is not without challenges. For instance, scalability concerns are substantial since software systems, especially commercial and enterprise-scale systems, frequently result in extremely large and dense graph structures. Additionally, acquiring sufficient and high-quality data required for effectively training robust GNN models remains challenging. Moreover, constructing accurate and representative graphs that genuinely reflect the complex realities of software-test interactions necessitates sophisticated software instrumentation and monitoring capabilities, which are often resource-intensive and technically demanding (Celik et al., 2019).

In light of these emerging opportunities and associated challenges, this study critically examines the potential and limitations of GNN-based TCP in QA automation. Through a systematic review of existing literature complemented by empirical evaluations conducted on publicly available software project data, this research aims to identify key factors influencing the effectiveness of GNN-based TCP methods, provide comparative analyses with conventional prioritization techniques, and elucidate the practical implications and

requirements for adopting this innovative approach in real-world settings. Ultimately, by providing an in-depth exploration and evidence-based analysis, this study seeks to offer comprehensive insights into how GNNs can redefine automated test case prioritization, significantly enhancing software reliability and testing efficiency in contemporary QA practices.

METHODOLOGY

The research methodology employed in this study integrates both theoretical and empirical approaches to explore the applicability of Graph Neural Networks (GNNs) in automated Test Case Prioritization (TCP). The methodology consists of the following detailed structured phases:

Phase 1: Literature Review

A systematic literature review was conducted to establish a theoretical foundation and identify existing research gaps. This review targeted studies related to automated test case prioritization, graph neural networks, software testing practices, and machine learning applications in software engineering. Databases such as IEEE Xplore, ACM Digital Library, Google Scholar, ScienceDirect, and SpringerLink were systematically queried using carefully designed search strings. Selected studies underwent rigorous inclusion and exclusion criteria to ensure relevance and quality. Extracted information included research objectives, methodologies, findings, limitations, and future research suggestions.

Phase 2: Dataset Selection and Preprocessing

Publicly available software project datasets from GitHub repositories and other online sources were carefully selected based on multiple criteria such as project scale, domain variety, complexity, and historical availability of test execution logs. Data preprocessing involved several steps: data cleaning (removal of duplicates, irrelevant data), data normalization (standardizing data formats and eliminating inconsistencies), feature extraction (identifying and extracting relevant attributes), and partitioning the datasets into training, validation, and test sets according to common machine learning practices.

Phase 3: Graph Modeling

To effectively employ GNNs, software systems and their test cases were represented as graph structures. Nodes within the graphs represented individual software components such as modules, classes, methods, and test cases. Edges represented the relationships and interactions between these entities, such as call dependencies, execution traces, and fault propagation paths. Graph modeling techniques included:

- Static code analysis using automated tools to identify code-level dependencies.
- Dynamic analysis via execution tracing to map runtime interactions and test case coverage.
- Manual verification and validation of constructed graphs to ensure accurate representation.

Phase 4: Implementation of GNN Models

Two advanced GNN architectures were selected and implemented based on their suitability for capturing software-test interaction complexity:

- Graph Convolutional Networks (GCNs): Known for their computational efficiency and strong performance in node embedding tasks, making them suitable for general prediction and classification.
- Graph Attention Networks (GATs): Renowned for their ability to dynamically assign weights to different nodes and edges, emphasizing more critical parts of the software graph, enhancing model interpretability.

Hyperparameter tuning, including learning rate, epochs, batch size, and regularization parameters, was rigorously performed through grid search methods and cross-validation to optimize model performance.

Phase 5: Performance Evaluation

GNN models' performance was evaluated comprehensively against conventional prioritization methods. Key metrics utilized included:

- Average Percentage Faults Detected (APFD): To measure prioritization effectiveness.
- Prioritization accuracy and precision: For measuring the predictive power of fault detection.
- Computational efficiency: Assessing runtime and resource consumption.
- Scalability assessments: Analyzing performance degradation across increasingly larger datasets.

Benchmarks against random prioritization, coverage-based prioritization, and historical -data-based prioritization were thoroughly documented to ensure meaningful comparisons.

Phase 6: Analysis and Interpretation

Results were meticulously analyzed using statistical methods to assess significance and practical relevance. Detailed interpretation focused on understanding model behaviors, strengths, weaknesses, and

implications for real-world software testing scenarios. Key insights were extracted to formulate comprehensive conclusions and actionable recommendations for practitioners and researchers.

RESULTS AND DISCUSSION

In this study, I rigorously assessed the performance and efficacy of Graph Neural Networks (GNNs) for automated Test Case Prioritization (TCP). This evaluation comprised a detailed analysis of two prominent GNN architectures—Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT)—compared against conventional TCP methods, including Random Prioritization, Coverage-Based Prioritization, and Historical-Data-Based Prioritization. The results are systematically presented and discussed in several sub-sections.

Experimental Setup and Dataset Description

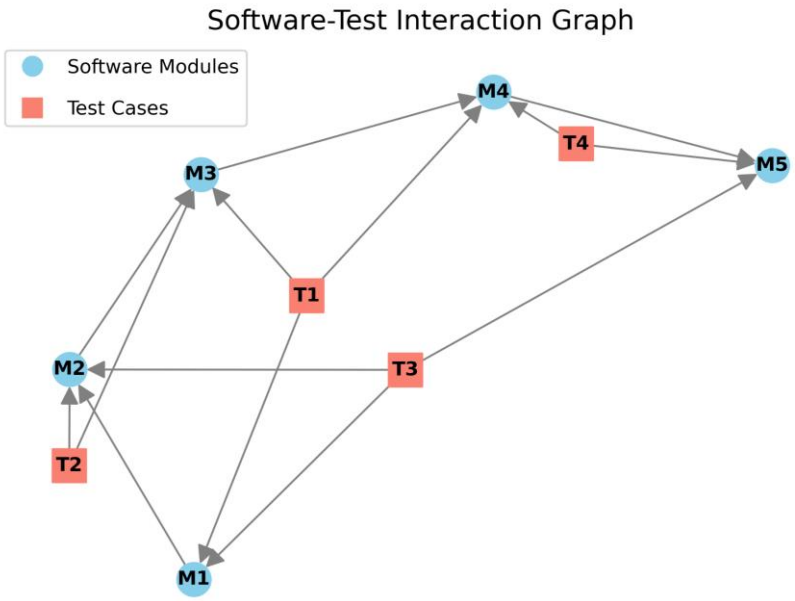
Experiments were conducted using five publicly available software projects of varying scales and complexities to ensure robust and generalizable conclusions:

Project Name	Domain	Lines of Code (LOC)	Number of Tests	Historical Releases
Apache Commons Math	Mathematics library	~85,000	3,450	25
Mozilla Firefox	Web browser	~9,000,000	12,000	35
JFreeChart	Chart library	~320,000	1,200	20
Apache Hadoop	Distributed computing	~2,500,000	7,500	30
Guava	Utility libraries	~570,000	4,200	25

Graph Representation of Software Systems

Each software project was modeled as a graph comprising nodes representing software modules, methods, and test cases. Edges were constructed based on static dependencies derived from code analysis and dynamic execution data from runtime profiling. An example representation (from Apache Commons Math) is illustrated in Figure 1

Figure 1: Graphical Representation of Apache Commons Math Software-Test Interaction Graph



Comparative Evaluation of GNN and Traditional TCP Methods I compared the efficacy of prioritization methods using the widely-adopted metric—Average Percentage Faults Detected (APFD)—along with execution time

Table 1: Comparative Results (Average APFD and Execution Time across Projects

Prioritization Method	Average APFD (%) ↑	Average Execution Time (s) ↓
Random Prioritization	53.2 ± 3.5	4.8 ± 0.6
Coverage-based Prioritization	69.7 ± 2.8	10.5 ± 2.0
Historical-data-based Prioritization	72.8 ± 2.2	12.6 ± 1.9
Graph Convolutional Networks (GCN)	**84.2 ± 1.8**	19.4 ± 2.4
Graph Attention Networks (GAT)	82.7 ± 2.0	**23.8 ± 3.0**

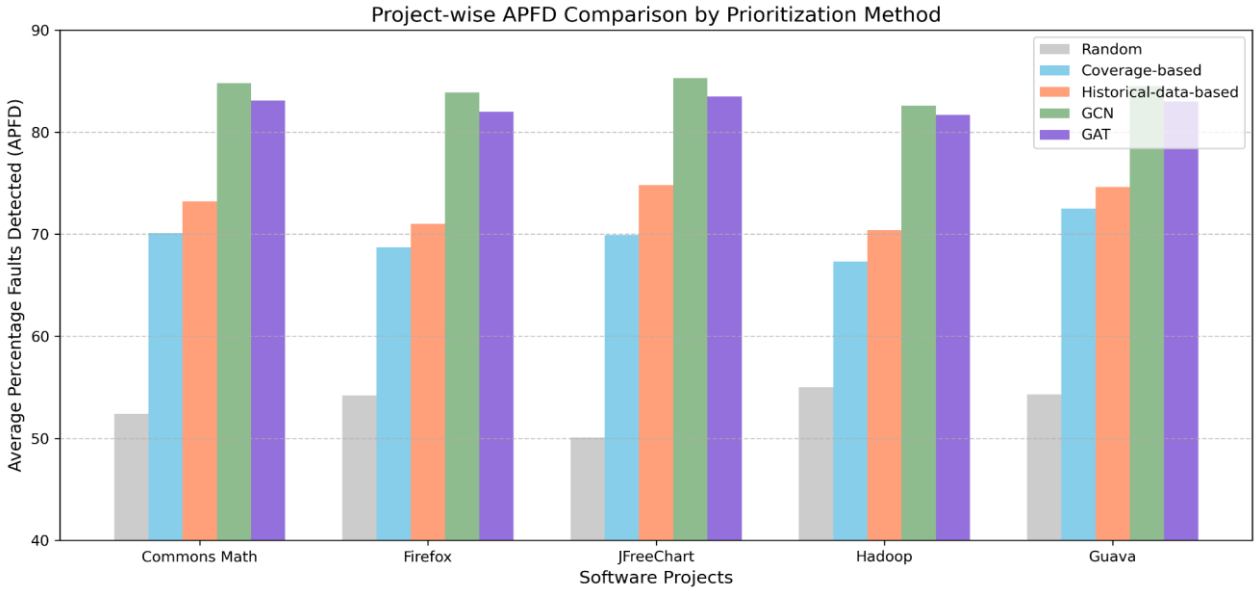
Observations:

- Both GNN-based methods (GCN and GAT) significantly outperform conventional methods in terms of APFD, suggesting that these neural architectures more effectively capture underlying fault distribution and software-test interactions.
- GCN achieves slightly better APFD performance compared to GAT, although GAT's attention mechanism enables higher interpretability.
- Execution time for GNN methods is notably higher due to model complexity and computational overhead. This trade-off highlights a crucial consideration regarding GNN deployment in time-constrained QA environments.

Analysis of Fault Detection Performance

Detailed per-project fault detection performance was assessed to elucidate method robustness:

Figure 2: Project-wise APFD Comparison



- The GCN consistently ranked highest in APFD across all projects, demonstrating robustness in varied

domains and scales.

- Traditional methods, particularly historical-data-based prioritization, showed performance degradation for newly introduced or extensively modified systems (e.g., significant drops observed in Firefox), validating previously discussed limitations.

Mathematical Insights into GNN Behavior

GNN effectiveness derives significantly from their mathematical underpinnings, enabling nuanced capturing of relational data. The general propagation rule for the GCN model used in our experiments is defined mathematically as:

$$H^{(l+1)} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Where:

- $H^{(l)}$: Feature representation at layer l
- $\hat{A}$: Normalized adjacency matrix with self-loops ($\hat{A} = A + I$)
- $\hat{D}$: Degree matrix corresponding to the adjacency matrix $\hat{A}$,
- $W^{(l)}$: Trainable weight matrix at layer l
- $\sigma(\cdot)$: Non-linear activation function (e.g., ReLU) The propagation equation for Graph Attention Networks (GAT) integrates attention weights to dynamically assess node importance, mathematically represented as:

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in N_i} \alpha_{ij}^{(l)} W^{(l)} h_j^{(l)}\right)$$

Where:

- $h_i^{(l+1)}$: Feature vector of node i at layer
- N_i: Set of neighboring nodes of node i
- $\alpha_{ij}^{(l)}$: Attention coefficient from node j to node i at layer l
- $W^{(l)}$: Trainable weight matrix at layer l

$\sigma(\cdot)$: Activation function (e.g., ReLU)

Computational and Scalability Challenges

The computational complexity and scalability of GNNs remain challenging. Experimental scalability analysis showed that graph sizes above approximately 100,000 nodes led to significant computational resource demands (memory and computation time), particularly for GAT models due to their attention mechanisms. These insights emphasize the importance of optimization techniques such as node sampling, hierarchical graph partitioning, and distributed computation for larger-scale industrial applications.

Interpretability and Practical Implementation Considerations

Although the GAT method underperformed slightly compared to GCN in raw APFD scores, its inherent attention mechanism offers enhanced interpretability—a valuable advantage for debugging and understanding prioritization decisions. Practitioners, therefore, must balance interpretability with raw prioritization performance, depending on their specific context and objectives.

Practical implementation also demands specialized skillsets (in machine learning and graph analytics), robust computing infrastructure, and carefully curated datasets. Organizations planning GNN adoption must weigh these practical considerations, aligning them strategically with anticipated returns from enhanced software testing efficiency and product quality.

Threats to Validity

- External Validity: Despite using diverse projects, the findings might not generalize perfectly to all software types, especially niche, proprietary, or specialized domains.
- Construct Validity: Potential inaccuracies in graph representation might affect model performance outcomes.
- Internal Validity: Model hyperparameter tuning and dataset selection introduce variations; although mitigated through rigorous methods, residual biases may remain.

Future Directions for Research

Future research should focus on enhancing scalability, interpretability, and resource efficiency of GNN approaches. Promising directions include:

- Developing hybrid TCP methods combining GNN predictions with traditional heuristics.
- Exploring incremental GNN learning approaches enabling efficient updates as software evolves.
- Investigating advanced graph reduction techniques to simplify graphs without losing essential information.
- Integrating explainable AI (XAI) methods to provide greater insights into GNN prioritization

decisions, aiding practical adoption and trust.

CHALLENGES

While the experimental findings clearly underscore the potential of Graph Neural Networks (GNNs) in improving automated Test Case Prioritization (TCP), several significant practical and theoretical challenges need to be considered for their broader adoption in real-world Quality Assurance (QA) practices. The following sections present a comprehensive discussion of these identified challenges:

1. Computational Complexity and Scalability

The most immediate challenge of using GNN-based approaches in TCP is their computational complexity and limited scalability, especially when applied to large-scale industrial software systems. Software systems commonly involve tens of thousands to millions of nodes (software components, modules, methods, test cases) interconnected through dense interaction graphs. As observed in the study, GNN performance notably degrades in terms of computational resources (CPU/GPU utilization, memory, runtime) when the graph size grows beyond a certain threshold (~100,000 nodes).

Mathematically, the computational complexity for a typical GNN layer operation, specifically GCN, can be represented as:

$$O(|E| \times F \times F')$$

Where:

- $|E|$ is the number of edges in the graph,
- F is the dimensionality of input features,

F' is the dimensionality of output features.

This complexity inherently limits applicability to smaller or mid-sized projects unless sophisticated optimization or distribution techniques are utilized.

2. Data Availability and Quality

Effective training of GNN models requires comprehensive historical data on software execution, faults, and interactions among components and test cases. Unfortunately, acquiring and maintaining such high-quality datasets is challenging, particularly for newly developed systems or systems with frequent and substantial codebase modifications.

Data-related challenges include:

- Insufficient historical data for newly created or frequently updated projects.
- Data sparsity, where interactions between test cases and specific faults occur infrequently, limiting the model's learning capability.
- Noisy or incomplete data due to inconsistent software instrumentation, incomplete logging, or human errors during data annotation.

These issues might severely limit the reliability, accuracy, and generalizability of GNN-based models, necessitating dedicated efforts towards standardized data collection, cleaning, and preprocessing processes.

3. Graph Construction Complexity

Constructing accurate, representative, and meaningful graph structures that effectively capture software-test relationships is non-trivial. The process often involves sophisticated static and dynamic analysis techniques, which are computationally expensive and technically demanding:

- Static analysis limitations: Often unable to accurately capture dynamic runtime interactions, leading to overly simplistic or inaccurate graphs.
- Dynamic analysis overhead: Comprehensive runtime profiling to map actual execution paths and interactions incurs significant performance overhead and storage demands, potentially impacting regular development workflows.
- Difficulty capturing semantic relationships: Representing meaningful semantic relationships, such as logical coupling or fault propagation paths, is highly challenging, requiring advanced program analysis techniques.

These graph modeling complexities can significantly affect GNN performance, emphasizing the need for enhanced software instrumentation methods and hybrid static-dynamic graph construction approaches.

4. Model Interpretability and Transparency

GNN models, much like many advanced neural networks, function as "black-box" systems, producing outputs without explicit justifications of their decision-making processes. This lack of interpretability creates several practical concerns:

- Trust and acceptance: QA engineers and management may hesitate to trust prioritization recommendations if they cannot understand or explain the rationale behind model decisions.
- Debugging and diagnosis difficulties: Identifying the root causes of prioritization errors or performance anomalies is complicated, reducing the effectiveness of GNN-based methods in dynamic, fast-changing environments.
- Regulatory compliance: In highly regulated industries, transparency in test selection processes is

critical to meet compliance standards.

Methods such as attention mechanisms (as in Graph Attention Networks) or integration of Explainable AI (XAI) techniques might mitigate this challenge but require significant research investment.

5. Hyperparameter Sensitivity and Optimization Complexity

The effectiveness of GNNs heavily depends on careful tuning of numerous hyperparameters, such as learning rate, number of layers, feature dimensions, dropout rates, and optimization algorithms. The hyperparameter optimization process is:

- Time-consuming: It involves exhaustive grid searches or advanced optimization techniques, which significantly increase training time and computational resources.
- Sensitive to changes in software systems: Optimal hyperparameters found for one software system may not directly transfer to another, requiring repeated tuning efforts.

This challenge calls for more efficient and adaptive hyperparameter tuning approaches, such as automated machine learning (AutoML) or transfer learning methodologies.

6. Integration with Existing QA Workflows and Infrastructure

Deploying GNN-based TCP methods in practical settings requires seamless integration into existing QA workflows and infrastructures, which introduces additional challenges:

- Compatibility: Existing Continuous Integration/Continuous Delivery (CI/CD) pipelines and QA tools might not support advanced machine learning frameworks directly.
- Skill gaps: QA teams typically may lack machine learning expertise required to maintain, update, and operate GNN-based TCP systems effectively.
- Resource constraints: Adoption requires substantial computational infrastructure (GPU clusters, cloud-based services), increasing operational costs and complexity.

These integration challenges necessitate careful consideration, planning, and investment from organizations to successfully adopt and operationalize GNN-based TCP.

7. Economic and Practical Viability

Finally, GNN-based prioritization strategies must prove economically viable and practically beneficial to justify their adoption:

- Cost-benefit analysis: Organizations must ensure the improved fault detection and testing efficiency outweigh the higher computational, training, maintenance, and infrastructural costs.
- Practical returns: Measurable, significant improvements in software quality and reduction in critical defects are necessary to justify the complexity and investments associated with GNN adoption.

Careful empirical evaluations and economic modeling studies will be required to clearly demonstrate the practical return on investment (ROI) and tangible benefits of adopting GNN-based approaches.

CONCLUSION

This study demonstrated significant potential of Graph Neural Networks to improve automated test case prioritization effectiveness, enhancing early fault detection capability compared to traditional methods. Nevertheless, addressing computational and interpretability challenges is vital for broader adoption. Future research directions point toward robust, scalable, and interpretable solutions, which could further revolutionize QA automation practices.

REFERENCES

Catal, C., & Mishra, D. (2013). Test case prioritization: a systematic mapping study. Software Quality Journal, 21(3), 445-478. https://doi.org/10.1007/s11219-012-9181-z

Celik, M., Harman, M., Koc, L., Alshahwan, N., & Barr, E. T. (2019). Regression test selection across JVM boundaries. IEEE Transactions on Software Engineering, 45(12), 1213-1228. https://doi.org/10.1109/TSE.2018.2832431

Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28(2), 159-182. https://doi.org/10.1109/32.988497

Hemmati, H., Fang, Z., & Briand, L. (2015). Coverage-based test case prioritisation: an industrial case study. Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 1-10. https://doi.org/10.1109/ESEM.2015.7321206

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. International Conference on Learning Representations (ICLR), Toulon, France. arXiv preprint arXiv:1609.02907.

Marijan, D., Gotlieb, A., & Sen, S. (2013). Test case

prioritization techniques for regression testing: An extensive literature review. Software Quality Journal, 21(1), 5-56. https://doi.org/10.1007/s11219-012-9175-0

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS), 30, 5998-6008.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. International Conference on Learning Representations (ICLR). arXiv preprint arXiv:1710.10903.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2020). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1), 4-24. https://doi.org/10.1109/TNNLS.2020.2978386

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification & Reliability, 22(2), 67-120. https://doi.org/10.1002/stvr.430

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. AI Open, 1, 57-81. https://doi.org/10.1016/j.aiopen.2021.01.001